PI Interface for Emerson DeltaV Batch

Version 3.0.3.x

OSIsoft, LLC
777 Davis St., Suite 250
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Fax: (01) 510-357-8136
Web: http://www.osisoft.com

OSIsoft Australia • Perth, Australia
OSIsoft Europe GmbH • Frankfurt, Germany
OSIsoft Asia Pte Ltd. • Singapore
OSIsoft Canada ULC • Montreal & Calgary, Canada
OSIsoft, LLC Representative Office • Shanghai, People's Republic of China
OSIsoft Japan KK • Tokyo, Japan
OSIsoft Mexico S. De R.L. De C.V. • Mexico City, Mexico
OSIsoft do Brasil Sistemas Ltda. • Sao Paulo, Brazil
OSIsoft France EURL • Paris, France

PI Interface for Emerson DeltaV Batch

Copyright: © 2008-2013 OSIsoft, LLC. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, photocopying, recording, or otherwise, without the prior written permission of OSIsoft, LLC.

OSIsoft, the OSIsoft logo and logotype, PI Analytics, PI ProcessBook, PI DataLink, ProcessPoint, PI Asset Framework (PI AF), IT Monitor, MCN Health Monitor, PI System, PI ActiveView, PI ACE, PI AlarmView, PI BatchView, PI Coresight, PI Data Services, PI Event Frames, PI Manual Logger, PI ProfileView, PI WebParts, ProTRAQ, RLINK, RtAnalytics, RtBaseline, RtPortal, RtPM, RtReports and RtWebParts are all trademarks of OSIsoft, LLC. All other trademarks or trade names used herein are the property of their respective owners.

U.S. GOVERNMENT RIGHTS

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the OSIsoft, LLC license agreement and as provided in DFARS 227.7202, DFARS 252.227-7013, FAR 12.212, FAR 52.227, as applicable. OSIsoft, LLC.

Published: 04/2013

Table of Contents

Chapter 1. Introduction 1

Reference Manuals 2
Supported Features 2
Diagram of Hardware Connection 6

Chapter 2. Principles of Operation 13

Interface Modes 13
Source Template 14
Event Journals as Data Source 17
SQL Batch Historian as Data Source 17
SQL Alarm&Events Historian as Data Source 18
OPC Alarm&Events Server as Data Source 19
Recipe Model vs. Equipment Model 19
PI Batch Database Methodology 21
PIBatch 22
PIUnitBatch 24
PISubBatches 28
Operation 28
Phase 30
Phase State 32
Phase Step 32
Arbitration Events Unavailable 33
Template Placeholders 34
PIBatch and PIUnitBatch Product Property 35
PIModule Creation 36
PI AF Event Frames Methodology 37
Procedure 39
UnitProcedure 41
Operation 45
Phase 48
Phase State 50
Phase Step 51
Arbitration Events Unavailable 52
PI AF Element Creation 52
Foreign Language Support 53
Template Placeholders 58
Event Logging 59
Advanced Parsing Parameters 61
Attribute / Property Templates 62
Tag Templates 68
Tag Templates – PI Batch Database Activity Logging 80
Alarm Tag Templates 82
PI Tag as Placeholder 86
Recipe Templates 87
Merging Multiple Source batches into a Single PIBatch 93

Using /BIDM Parameter 94
Linking BES to MES batches 94
Lost Connections to PI Server and PI Archive Backup Issues 95
Data Preprocessing 95
Data Recovery 97
Data Analysis 98
PI Data Deletion 99
EVT Source – Event Based Time Ordered Processing 99
Dealing with Irrelevant Recipes 100
Dealing with Irrelevant Units 101
Dealing with Irrelevant Phases 101
Dealing with Irrelevant Phase States 102
Initialization File 102
EVT Source – Example Event File Journal 104

Chapter 3. Installation Checklist 107

Data Collection Steps 107
Interface Diagnostics 108

Chapter 4. Interface Installation 113

Naming Conventions and Requirements 113
Interface Directories 114
PIHOME Directory Tree 114
Interface Installation Directory 114
Interface Installation Procedure 114
Installing the Interface as a Windows Service 114
Installing the Interface Service with the PI Event Frames Interface Manager 114
Service Tab 115
Installing the Interface Service Manually 115

Chapter 5. Digital States 117

Chapter 6. PointSource 119

Chapter 7. PI Point Configuration 121

Interface-specific Points 121

Chapter 8. Startup Command File 123

Configuring the Interface with PI Event Frames Interface Manager 123
Interface Selection Tab 123
File Selection Tab 123
Server Information Tab 124
PI Server (/HOST) 124
[PI Server] User and [PI Server] Password 124
[PI Server] Port 124
Use PI AF server 124
[PI AF] Host and Database (/AFHOST and /AFDATABASE) 124
[PI AF] User and [PI AF] Password 124
Source 125

SQL Server 125
SQL Database 125
User and Password....125
Use Alarms and Events Historian....125
DeltaV OPCAE OPC Node and OPC Server 125
Filters Tab 125
Skip Phases (/SKIPPHASES)....125
Skip Units (/SKIPUNITS) 125
Skip Recipes (/SKIPRECIPES) 126
Exclude Phase States (/EXCLUDESTATES) 126
Time Settings Tab 126
Query Time Settings 126
PI Connection Settings 127
SQL server Connection Settings 127
Operational Settings Tab 127
Runtime mode (/MODE=<mode>)....128
Perform one scan then stop (/SINGLERUN) 128
Print result of first scan to file (/PRINT=<file name>) 128
Debug level (/DB=<#>) 128
Numeric settings (/NS=<lang>) 128
Interface ID (/ID=x) 128
Point source (/PS=x)....129
Associate all reference elements with child Event Frames (/DPRETC)129
Failover Settings 129
Security Settings....129
Save Settings Tab....129
Test Configuration Tab....129
Configuring Interface Startup Files....130
Command-line Parameters 130
Sample PIEMDVB.bat File 142
Initialization File Parameters 142
Sample INI file – Multiple EVT Sources 145
Sample INI file – DeltaV German EVT Source 146
Sample INI file – DeltaV SQL....147
Sample INI file – DeltaV SQL, OPCAE 148

Chapter 9. Interface Node Clock....149

Chapter 10. Security....151

Chapter 11. Starting and Stopping the Interface 153

Starting Interface as a Service....153
Stopping the Interface Running as a Service....153

Chapter 12. Failover 155

Appendix A. Error and Informational Messages....157

Message Logs....157
Messages....157
System Errors and PI Errors 164

Appendix B. Batch Executive System – Configuration Requirements 165

Introduction 165
Background 165
Objectives 165
Principles of Operation 165
Principles of the PI Server Batch Database 165
Principles of the PI Interface to the Emerson DeltaV Batch 166
Recommendations for BES Recipes and Equipment Models 167

Appendix C. Event File Directory Sync Utility 171

Introduction 171
Principles of Operation 171
Utility Installation Procedure 171
Installing the Utility as a Windows Service 172
Startup Command File 172
Command-line Parameters 172
Sample EVTSync.bat File 172
Starting and Stopping the Utility 173
Starting the Utility Service 173
Stopping the Utility Service 173
Conclusions 173

Appendix D. Technical Support and Resources 175

Before You Call or Write for Help 175
Help Desk and Telephone Support 175
Search Support 176
Email-based Technical Support 176
Online Technical Support 176
Remote Access 177
On-site Service 177
Knowledge Center 177
Upgrades 177

Appendix E. Revision History 179

Terminology

To understand this interface, you should be familiar with the terminology used in this manual.

Interface Node

An Interface Node is a computer on which

- the PI API, the PI SDK, or both are installed, and
- PI Server programs are not installed.

PI API

The PI API is a library of functions that allow applications to communicate and to exchange data with the PI Server.

PI Collective

A PI Collective is two or more replicated PI Servers that collect data concurrently. Collectives are part of the High Availability environment. When the primary PI Server in a collective becomes unavailable, a secondary collective member node seamlessly continues to collect and provide data access to your PI clients.

PIHOME

`PIHOME` refers to the directory that is the common location for PI 32-bit client applications.

On a 32-bit operating system

A typical `PIHOME is C:\Program Files\PIPC.`

On a 64-bit operating system

A typical `PIHOME is C:\Program Files (x86)\PIPC.`

PI interfaces reside in a subdirectory of the `Interfaces` directory under `PIHOME`.

For example, files for the Modbus Ethernet Interface are in

`[PIHOME]\PIPC\Interfaces\ModbusE.`

This document uses `[PIHOME]` as an abbreviation for the complete `PIHOME` or `PIHOME64` directory. For example, ICU files in `[PIHOME]\ICU`.

PIHOME64

`PIHOME64` will be found only on a 64-bit operating system and refers to the directory that is the common location for PI 64-bit client applications.

A typical `PIHOME64 is C:\Program Files\PIPC`.

PI interfaces reside in a subdirectory of the `Interfaces` directory under `PIHOME64`.

For example, files for a 64-bit Modbus Ethernet Interface would be found in

`C:\Program Files\PIPC\Interfaces\ModbusE.`

This document uses [PIHOME] as an abbreviation for the complete PIHOME or PIHOME64 directory. For example, ICU files in [PIHOME]\ICU.

PI SDK

The PI SDK is a library of functions that allow applications to communicate and to exchange data with the PI Server. Some PI interfaces, in addition to using the PI API, require the PI SDK.

AF SDK

The AF SDK is a library of functions that enable applications to communicate and to exchange data with the AF Server. Some PI interfaces, in addition to using the PI API, PI-SDK, require the AF SDK.

PI Server Node

A PI Server Node is a computer on which PI Server programs are installed. The PI Server runs on the PI Server Node.

PI SMT

PI SMT refers to PI System Management Tools. PI SMT is the program you use for configuring PI Servers. A single copy of PI SMT manages multiple PI Servers. PI SMT runs on either a PI Server Node or a PI Interface Node.

Pipc.log

The pipc.log file is the file to which OSIsoft applications write informational and error messages. While a PI interface runs, it writes to the pipc.log file. The ICU provides easy access to the pipc.log.

Point

The PI point is the basic building block for controlling data flow to and from the PI Server. For a given timestamp, a PI point holds a single value.

A PI point does not necessarily correspond to a "data collection point" on the foreign device. For example, a single "point" on the foreign device can consist of a set point, a process value, an alarm limit, and a discrete value. These four pieces of information require four separate PI points.

Service

A Service is a Windows program that runs without user interaction. A Service continues to run after you have logged off as a Windows user. A Service has the ability to start up when the computer itself starts up.

The ICU allows you to configure a PI interface to run as a Service.

Tag (Input Tag and Output Tag)

The tag attribute of a PI point is the name of the PI point. There is a one-to-one correspondence between the name of a point and the point itself. Because of this relationship, PI System documentation uses the terms "tag" and "point" interchangeably.

Interfaces read values from a device and write these values to an Input Tag. Interfaces use an Output Tag to write a value to the device.

Chapter 1. Introduction

This manual describes the operation of the PI Interface to the Emerson DeltaV Batch. In this manual, we refer to the PI Interface to the Emerson DeltaV Batch as the Batch Interface. The primary objective of the Batch Interface is to collect batch processing events from the DeltaV System and store them in the PI Batch Database or PI AF Database (as event frames). In addition to collecting batch data, the interface collects associated batch data to PI Tags, and PI Batch properties or PI AF event frame attributes.

The Batch Interface is the first dedicated interface for collecting batch data from the DeltaV System. The interface can collect batch events in real-time through the DeltaV OPC Alarm & Events Server (A&E Server), which is optional for configuration. Associated batch data, such as operator comments, report parameters, and recipe parameters, are retrieved by querying the DeltaV Batch Historian during each interface scan. If you lose the connection to the DeltaV OPC A&E Server, the interface retrieves batch data and associated batch data from the DeltaV Batch Historian during each interface scan. The interface automatically tries to re-establish the connection to the DeltaV OPC A&E Server; once the connection has been re-established, the interface returns to collecting batch data in real-time through the DeltaV OPC A&E Server.

This interface is primarily designed to be used for DeltaV 10.3 and later systems utilizing the DeltaV OPC A&E Server and the DeltaV Batch Historian; however, it can run against earlier systems utilizing different data sources.

- For DeltaV 9.3 systems this interface can utilize the DeltaV Batch Historian or DeltaV event files as the primary data source.
- For DeltaV 8.4 systems this interface can only use DeltaV event files as the primary data source.

NOTE: The use of DeltaV event files as a public interface for the DeltaV System is not recommended by Emerson.

The flow of data in the interface is unidirectional—that is, data can only be read from the specified data source and written to the PI System. This interface can read data from multiple batch data sources simultaneously. By design, the interface does not edit or delete source data.

The Batch Interface is a scan-based interface that populates the AF Database (event frames and elements) or the PI Batch Database and PI Module Database. In addition to batch data, the interface can populate the PI Point Database. PI Point creation, commonly known as tag creation and event population, is controlled by using tag templates. All modules, tags, tag aliases, and health tags are automatically created on the PI server. The Interface does not use the PI API Buffering Service because batch and tag data is already buffered by the source historian databases. To maximize performance, the interface writes events to PI tags in bulk—that is, it writes all events per interface scan.

NOTE: The PI Interface to the Emerson DeltaV Batch is not an upgrade to the Batch Event File Monitor Interface. OSIsoft has provided information to customers on how to migrate from the Batch Event File Monitor (EVT) Interface to the PI Interface to the Emerson DeltaV Batch. This migration plan and best practices can be found on the OSIsoft Technical Support website in KB Article KB00430 (http://techsupport.osisoft.com/Support+Solution/8/KB00430.htm).

Reference Manuals

OSIsoft

- PI Data Archive Manual
- PI Server System Management Guide
- PI SDK User Manual
- AF SDK User Manual

Vendor

Review the pertinent documentation regarding the particular Batch Executive System (BES) at your facility. Maintain familiarity with the contents and format of the source data so that you can choose appropriate options and features for the interface.

Supported Features

Feature	Support	
Part Number	PI-IN-EM-DVB-NTI	
* Platforms	32-bit Interface	64-bit Interface
Windows XP		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows 2003 Server		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows Vista		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows 2008		
32-bit OS	Yes	No
Windows 2008 R2		
64-bit OS	Yes (Emulation Mode)	No
Windows 7		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows 8		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No

Feature	Support	
Auto Creates PI Points	Yes	
Point Builder Utility	No	
ICU Control	No (has own configuration tool)	
PI Point Types	Integer/ Float32/ String	
Sub-second Timestamps	Yes	
Sub-second Scan Classes	No	
Automatically Incorporates PI Point Attribute Changes	No	
Exception Reporting	No	
Outputs from PI	No	
Inputs to PI	Event and Scan-based	
Supports Questionable Bit	No	
Supports Multi-character PointSource	Yes	
Maximum Point Count	None	
* Uses PI SDK	Yes	
* Uses AF SDK	Yes	
PINet String Support	N/A	
* Source of Timestamps	Device	
History Recovery	Yes	
UniInt-based * Disconnected Startup * SetDeviceStatus	No No Yes	
Failover	Yes	
* Vendor Software Required on PI Interface Node/PINet Node	No	
* Vendor Software Required on Foreign Device	Yes	
* Vendor Hardware Required	No	
Additional PI Software Included with Interface	No	
Device Point Types	String/Integer/Float	
Serial-Based Interface	No	

*See paragraphs below for further explanation.

Platforms

The Interface is designed to run on the above mentioned Microsoft Windows operating systems. Because it is dependent on vendor software, newer platforms may not yet be supported. Please contact OSIsoft Technical Support for more information.

PI SDK

The PI SDK and the PI API are bundled and must be installed on each PI Interface node. The PI Interface to the Emerson DeltaV Batch makes PI SDK calls to access the PI Module Database and PI Batch Database. The Interface requires PI SDK version 1.3.4.333 or higher to be installed. The Interface uses PI API to log messages in the local pipc.log file. It does not require a PI API connection to the PI Server.

AF SDK

The AF SDK must be installed on each PI Interface node. The PI Interface to the Emerson DeltaV Batch makes AF SDK calls to access AF elements and AF event frames. The interface requires AF SDK version 2.5.x or higher to be installed prior the execution of the interface.

Source of Timestamps

Since each record in the source contains a timestamp and the interface itself is solely scan-based, use of the time at record processing could introduce inherent latency with respect to establishing the event time. Thus, the timestamp accompanying the record is used as the source of the timestamp for the data to be placed into the PI system. For the health tags, the Interface uses local system time at the time the value is being recorded.

History Recovery

The operation of the Batch Interface may be interrupted without loss of data. While the Interface is offline, the data is being buffered by the data sources such as SQL Server (DeltaV 9.3+), Event Journal files (DeltaV 8.4+).

The Interface can recover data provided it is still available in the data sources. If the data interruption occurred while the interface was running, then the data is recovered automatically without user intervention. To perform historical data recovery, the Interface must be run in **`Recovery`** mode. In this mode, the Interface can recover data for any time period specified by the user. The recovery mode is enabled by specifying the recovery time period through the command line parameters **`/rst=<date and time>`** (required) and **`/ret=<date and time>`** (optional). Note, the data recovery is limited by BES historical data availability as well as by few other factors on the PI Server, like the number of licensed tags, the size and time frame of PI archives into which data is backfilled, etc. Refer To Data Recovery section for more information.

SetDeviceStatus

The Health PIPoint with the attribute ExDesc = [UI_DEVSTAT], is used to represent the status of the source devices. This tag is automatically created and configured if missing by the interface on startup. The following events can be written into the tag:

a) "Good" – the interface is properly communicating and reading data from the data sources.

b) The following events represent proper communication with the data sources. This message is displayed on successful connection to each source.

"2 | Connected/No Data | EVT Directory Monitor: <directory name> Initialized."

"2 | Connected/No Data | Source SQL Server: <server name> Initialized."

c) The following list of events represents the failure to communicate with either the Event Journal file directory or Position directory, or failure to read data from the Event Journal File:

"3 | 1 device(s) in error | Error monitoring directory (onError): <directory name>"

"3 | 1 device(s) in error | Error monitoring directory: <directory name>"

"3 | 1 device(s) in error | Failed to start directory monitoring thread: <directory name>"

"3 | 1 device(s) in error | Error in scanning directory: <directory name>"

"3 | 1 device(s) in error | Error obtaining EVT files EOF."

"3 | 1 device(s) in error | Error getting current EVT file timestamp."

"3 | 1 device(s) in error | Error reading EVT file: <filename>."

"3 | 1 device(s) in error | Error while reading EVT file."

"3 | 1 device(s) in error | Error reading SQL Server: <server name>."

Vendor Software Required

The Batch Executive System (BES) and its accompanying support software are required for proper operation of this Batch interface.

When DeltaV SQL server is selected as data source, the SQL Native Client must be installed on the interface node.

Device Point Types

Since the interface receives data from source as string type, it attempts to coerce the string data into numerical equivalents according to Tag Templates if defined.

Diagram of Hardware Connection

Figure 1a. Schematic of Recommended Hardware and Software Configuration for Batch interface with event files as data source and PI Server as host, writing to PI Batch.

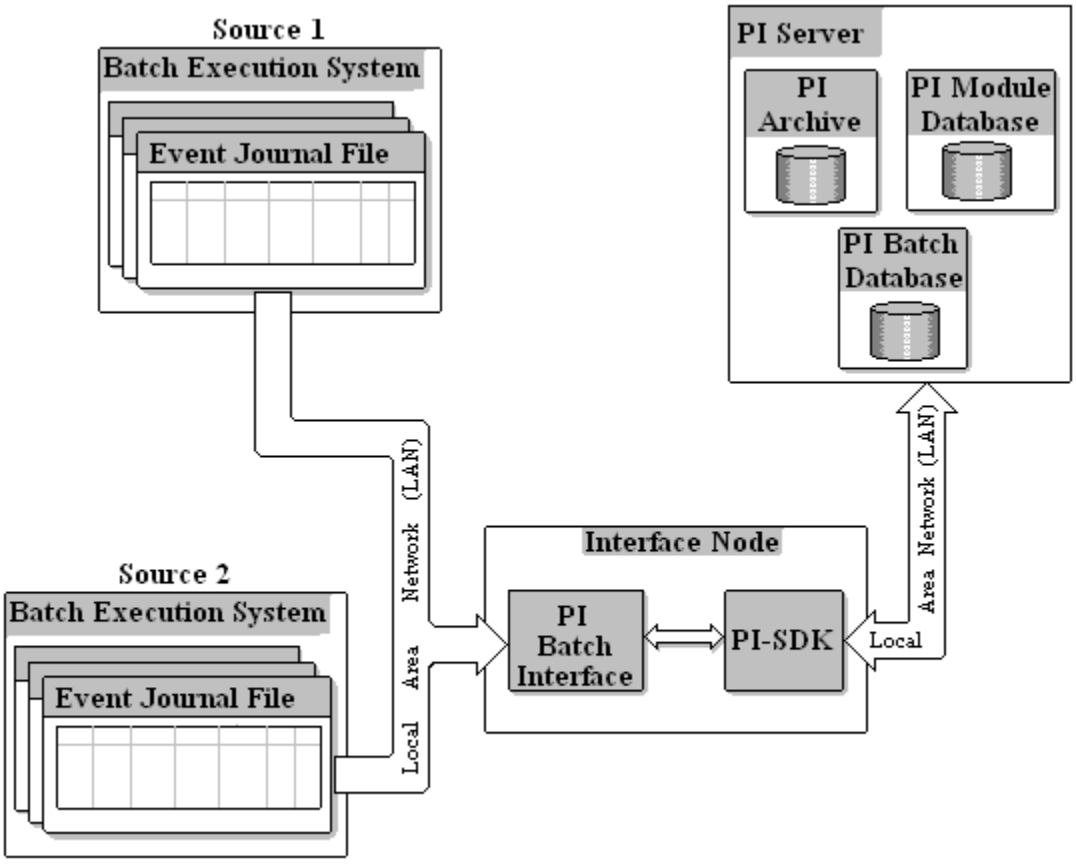


Figure 1b. Schematic of Recommended Hardware and Software Configuration for Batch interface with event files as data source and AF Server as host for asset and batch data.

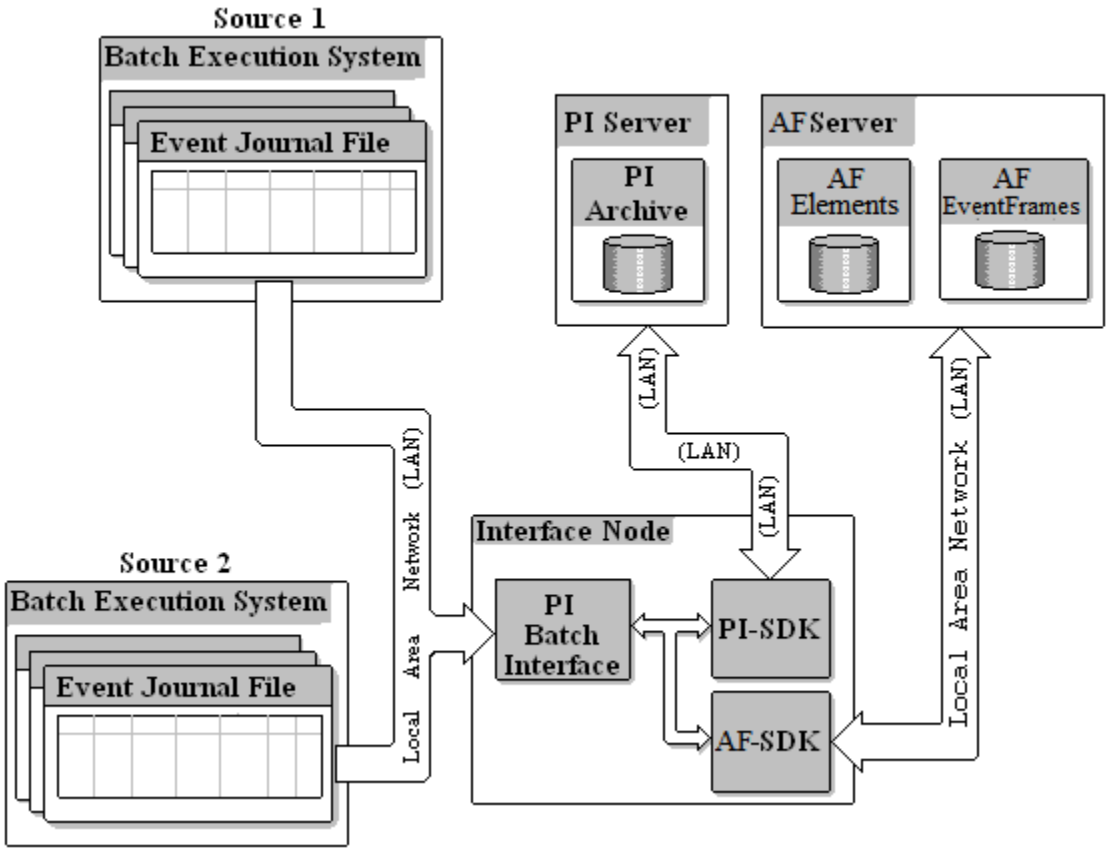


Figure 2a. Schematic of Recommended Hardware and Software Configuration for Batch interface with DeltaV SQL servers as data sources, writing to PI Batch.

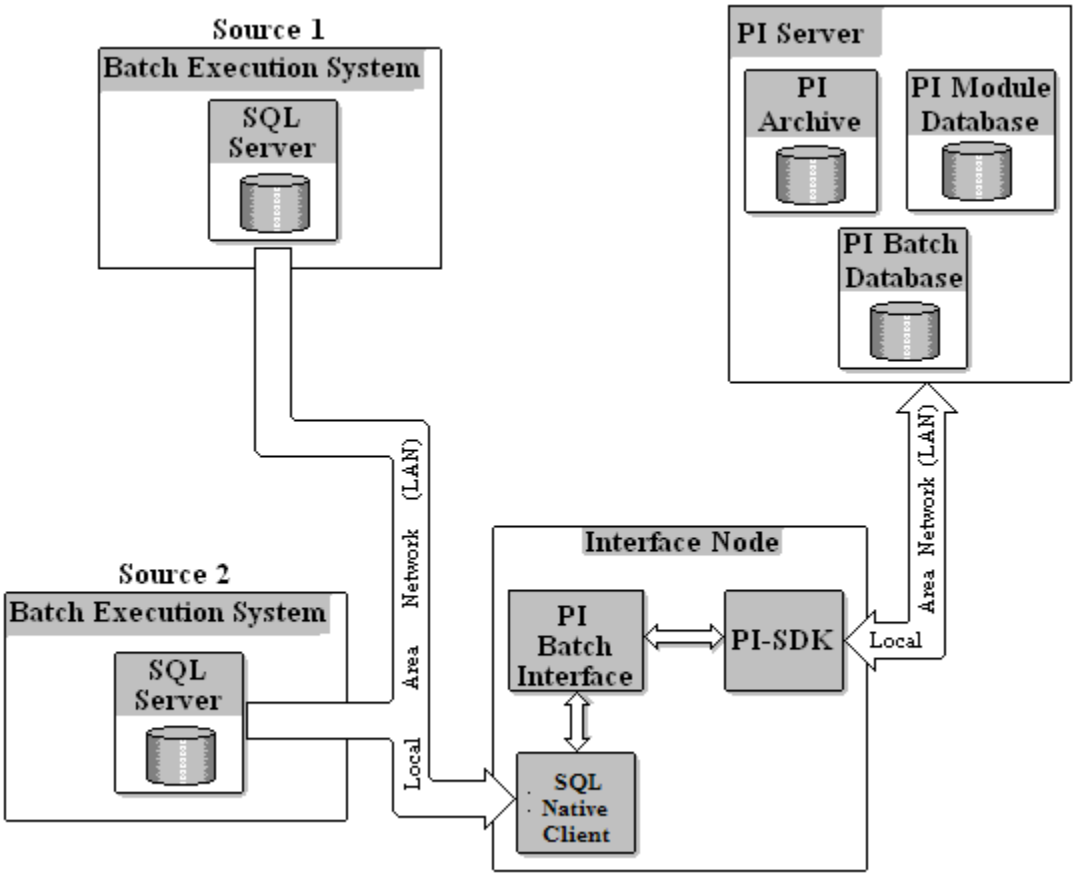


Figure 2b. Schematic of Recommended Hardware and Software Configuration for Batch interface with DeltaV SQL servers as data sources and the AF Server as host for asset and batch data.

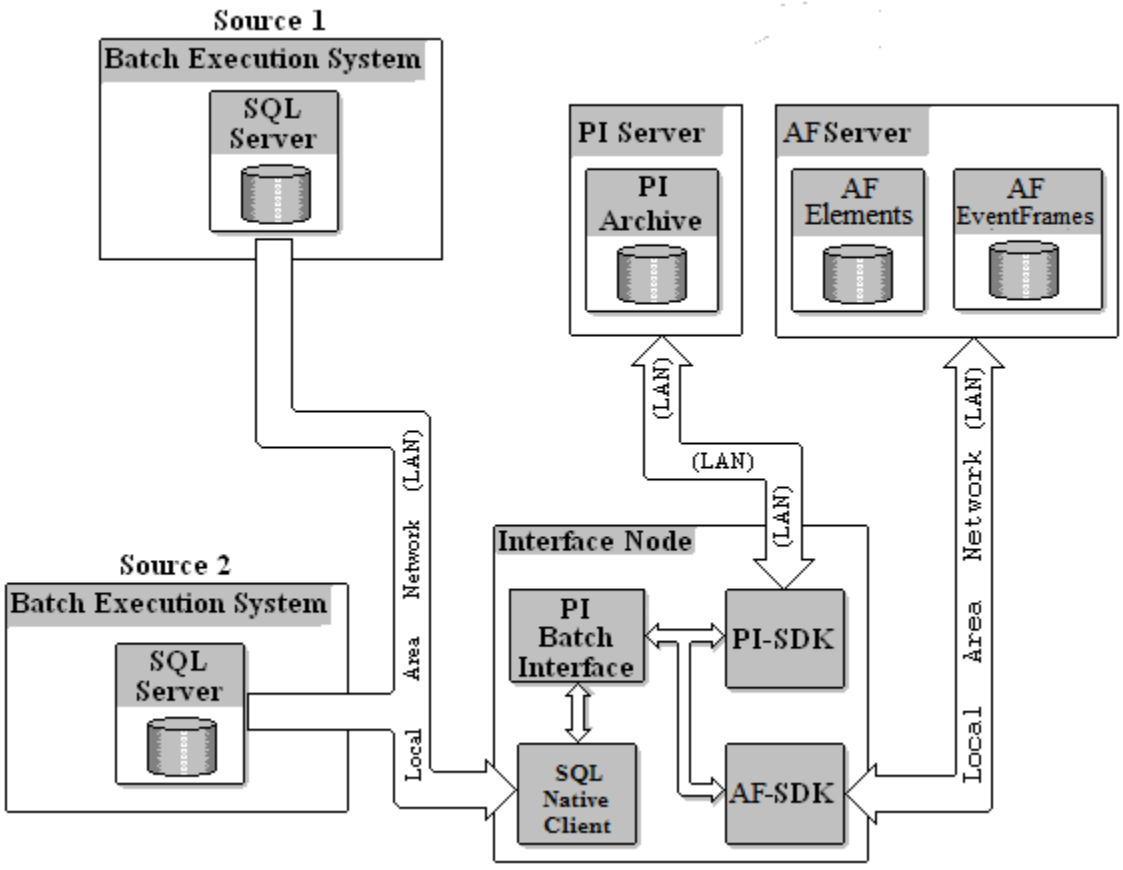


Figure 3a. Schematic of Recommended Hardware and Software Configuration for Batch interface with DeltaV SQL and OPC AE servers as data sources, writing to PI Batch.

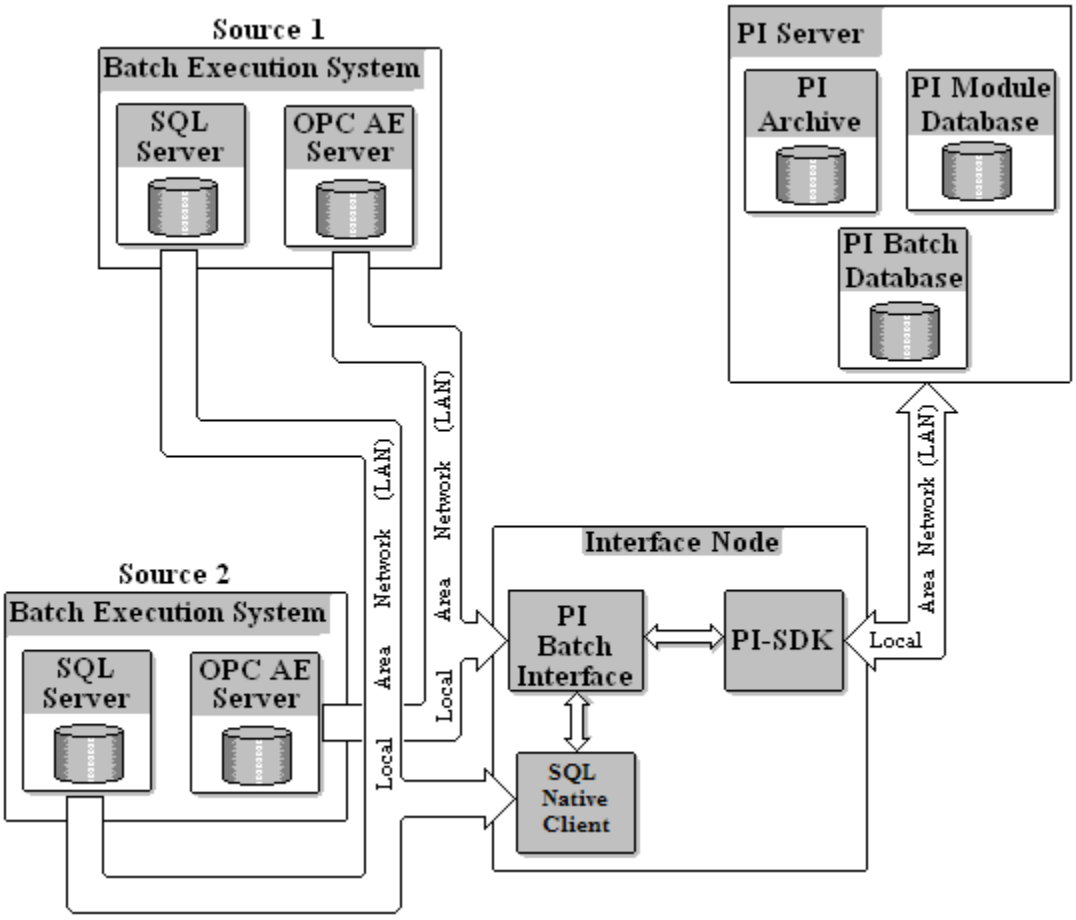
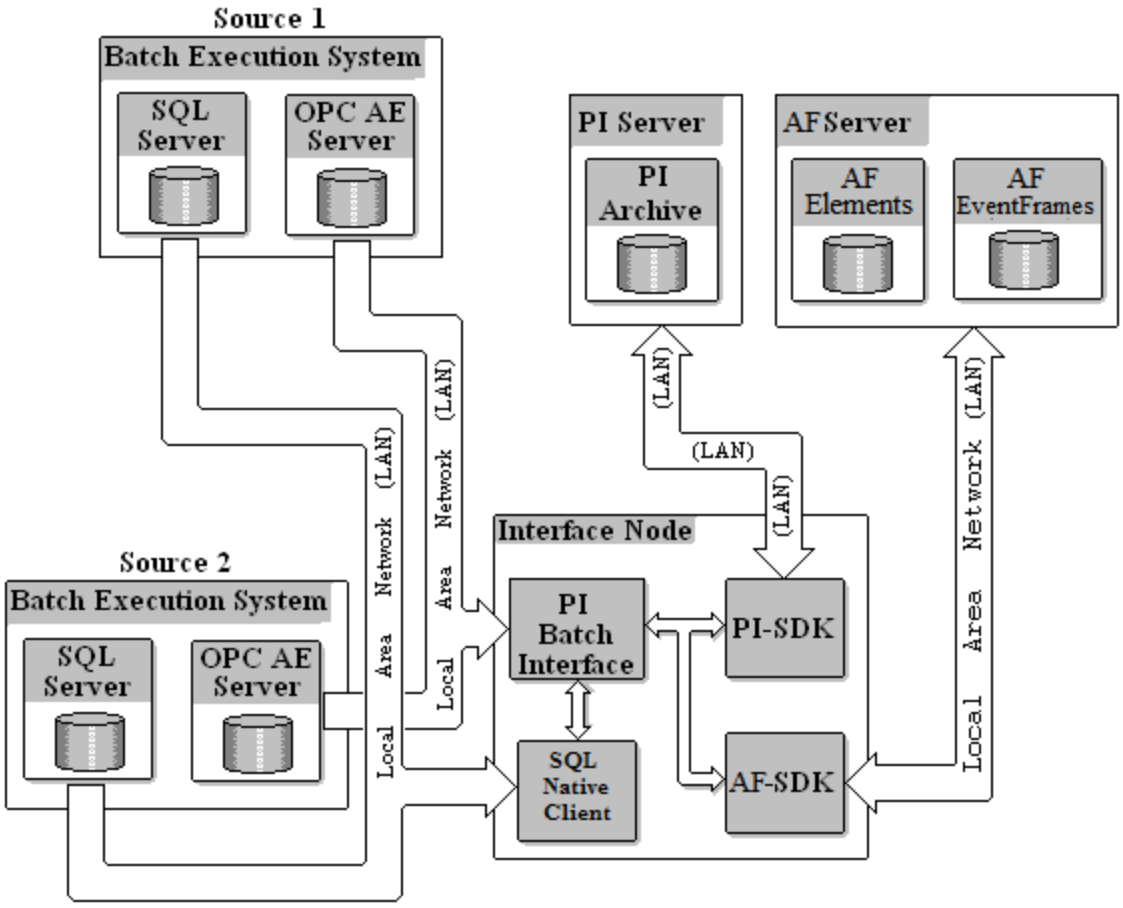


Figure 3b. Schematic of Recommended Hardware and Software Configuration for Batch interface with DeltaV SQL and OPC AE servers as data sources and AF server as host for asset and batch data.



The Batch interface can be installed on the same node as the batch execution system (BES) or on a completely separate node. Due to load-balancing considerations, OSIsoft does not recommend that the interface be installed on the same node as the PI Server. Contact the vendor of your BES for recommendations as to installing third-party software such as the PI Interface to the Emerson DeltaV Batch, on the same node as the Emerson DeltaV Batch Executive.

Chapter 2. Principles of Operation

This section contains relevant information to help the user better understand some of the primary logic of the Emerson DeltaV Batch interface.

Interface Modes

The Interface can be run in five different modes:

- RealTime (default)
- Recovery
- Preprocess
- Statistics
- Delete

RealTime mode is the default mode of operation and **Recovery** mode is designed to recover historical batch and tag data, provided the data still exists on the source. The principal difference between **RealTime** and **Recovery** modes is that in **RealTime** mode the interface synchronizes newly acquired data from the source with the PI Server at the end of each scan regardless of batch completion on the source. In **Recovery** mode, the interface synchronizes the batch only when it has completed on the source—that is, the end time is known.

In **Recovery** mode, all open batches are processed only when there are no completed batches left to be processed, when we have reached the current time. If the interface is started in **Recovery** mode without defining the Recovery End Time (interface command line parameter **/ret=<date and time>**), it prints the results of the recovery process and change to **RealTime** mode as soon as it reaches current time. The Recovery mode is always used on interface startup. The recovery is performed from the timestamp of the last processed event to the PI Server before shutdown until the interface reaches the current time. The mode is then automatically changed to the **Realtime**. **Recovery** mode can be also enabled through the use of the optional command line parameter – Recovery Start Time **(/rst=<date and time>)**. This parameter allows you to specify an alternative history recovery start time. The history recovery end time is optional and can be specified through the command line parameter – Recovery End Time **(/ret=<date and time>**). The Recovery End Time has no effect unless the **(/rst)** parameter is specified.

Note: If the Recovery End Time switch is used, the interface stops on recovery completion.

The **Preprocess** mode is designed for situations when the source data must be written to PI archives with earlier timestamps than the primary PI archive. Due to the nature of the PI Server, newly added tags, units and modules are indexed (referenced) only in the primary PI

archive. Any older archive will not have knowledge of these modules, units and tags. In **Preprocess** mode the interface creates only modules, units, tags and tag aliases without processing batch data and adding events into the tags. On completion, the interface stops and the user has to reprocess older archives with the offline archive utility. Please refer to the *PI Server System Management Guide* for details on archive reprocessing procedure (note that these procedures have changed with PI Server 2012). The reprocessing creates indexes for newly added units, modules, tags in each reprocessed archive. This mode should be always used before writing new batch data to older PI archives. It can be enabled by simply adding the **/mode=NoData** parameter to the command line parameters in conjunction with the Recovery Start Time switch **(/rst=<date and time>**. OSI does not recommend using the Recovery End Time **/ret=<date and time>** parameter because it can cause incomplete data processing, and therefore all tags and modules would not be created on the PI server.

In **Statistics** mode, the interface compares source data with the PI server data. In this mode the interface does not write or modify any data on the PI Server. Upon completion the interface reports results and stops. To enable this mode, the command line parameter **(/mode=stat)** has to be specified in command line parameters in conjunction with the Recovery Start Time parameter **(/rst=<date and time>)**. The Recovery End Time parameter **/ret=<date and time>)** can also be specified to limit the time frame of interest. If not specified, the interface will compare data from Recovery Start Time until current time.

In **Delete** mode, the interface cleans PI archives based on specified source data only, leaving data from all other sources intact. This mode should be used only if the interface is unable to synchronize source batch data with the PI server. This modes is used only in conjunction with Recovery mode command line parameters **(/rst** and **/ret**) and can be enabled by adding the parameter **(/mode=delete**) to the command line parameters in the interface startup file.

Source Template

The Batch interface supports simultaneous data processing coming from multiple sources. Primarily, parallel processing is designed for processing data from distributed control Batch Execution Systems. For example, the control logic of manufacturing process can be split among upstream and downstream segments, where each segment is controlled by a separate DeltaV Batch Executive. Even though the logical batch is the same, the actual batch related data is split among two batch historians. This Batch interface allows merging data for such batches and storing it into a single PI batch. Refer to section Merging Multiple Source batches into a Single PIBatch for more details. Parallel data processing resolves the problem with shared unit control as well, where different overlapping batch recipes can access the same unit in different stages of its production cycle. This is achieved through acquiring data for the same time frame from multiple sources and process combined time ordered data by the single instance of the interface. Data source(s) should be defined in the INI file associated with the specific interface. If EVT files are used as data sources, then ONLY the full path EVT directory should be defined for each source object. When DeltaV SQL historians are used as data sources, then the SQL server name and optionally database (default database: DVHisDB) should be defined for each source object. For Emerson DeltaV 10.3+ BES, this interface is able to retrieve batch data in realtime from DeltaV embedded OPC AE server. Since OPCAE server is a DA server, the interface requires that SQL to be used for data recovery. For cases when such mode of operation is desired, OPC AE node name and optional server name (default: DeltaV.OPCEventServer.1) should be defined in conjunction with SQL server name.

Table 1. Data source usage and description.

Object Name	Property name	Description
Source[#]		Defines the interface data source, where # is the index of the source. Index can be any positive 1-based number. The index is only used to match multiple source properties to the same source object, such as: opcnode, opcserver OR sqlserver, sqldatabase, sqluser , sqlpswd.
	.evtdir= [directory path] Required for EVT data Source	Defines the Event File journal directory associated with particular source. Example: Source[1].evtdir = D:\TEST\RELEASE\test_1 Source[2].evtdir = D:\TEST\RELEASE\test_2 Source[3].evtdir = D:\TEST\RELEASE\test_3
	.opcnode= [Node Name / IP] Required for OPCAE data Source Available in DeltaV 10.3+	Defines the name of the node (machine) where the DeltaV OPCAE server is installed. Required for OPCAE data collection. If used in conjunction with DeltaV SQL server, it must be defined under the same source. Example: Source[1].sqlserver = deltav10 Source[1].sqldatabase= DVHisDB Source[1].opcnode = deltav10 Source[1].opcserver = DeltaV.OPCEventServer.1 Source[2].sqlserver = deltav10 Source[2].opcnode = 192.168.1.10 (for Source 2: using default SQL database name and default OPCAE server name)
	.opcserver = [Server Name] Optional for OPCAE data Source Default: DeltaV.OPCEventServer.1 Available in DeltaV 10.3+	Defines the name of DeltaV OPCAE Server. Optional for OPCAE data collection. If used in conjunction with DeltaV SQL server, it must be defined under the same source. Default: DeltaV.OPCEventServer.1 Example: Source[1].opcnode = deltav10 Source[1].opcserver = DeltaV.OPCEventServer.1 When used with DeltaV SQL (same source): Source[1].sqlserver = deltav10 Source[1].sqldatabase = DVHisDB Source[1].opcnode = deltav10 Source[1].opcserver = DeltaV.OPCEventServer.1
	.sqlserver= [Server Name / IP] Required for SQL data Source Available in DeltaV 9.3+	Defines the name of the source DeltaV BES SQL server name. Example: source[1].sqlserver = 192.168.1.10 source[1].sqldatabase = PrimeDB source[2].sqlserver = deltav10 (for source 2: using default database: DVHisDB)

Object Name	Property name	Description
	.sqldatabase = [Primary Database Name] Optional for SQL data Source Default: DVHisDB Available in DeltaV 9.3+	Defines the name of the primary database name in DeltaV BES SQL server. Must be used in conjunction with property: .sqlserver=[Server Name] Example: source[2].sqlserver = deltav9 source[2].sqldatabase = DVHisDB
	Cursor = Client / Server Optional for SQL data source Default: Client Available in DeltaV 9.3+	Defines the SQL DataSet cursor location. Valid values: - Server: The interface requests and processes one dataset record at a time. It reduces interface node memory consumption - Client: The interface retrieves complete dataset prior to processing. High memory consumption on interface node. This is default mode.
	.sqluser=[SQL user name] Optional for SQL data Source Default: Windows authentication Available in DeltaV 9.3+	Defines the explicit user name used for connection to DeltaV SQL server. Example: source[1].sqlserver = deltav9 source[1].sqldatabase = DVHisDB source[1].sqluser = Johns
	.sqlpswd = [SQL user password] Optional for SQL data source Available in DeltaV 9.3+	Defines the user password used for connection to DeltaV SQL server. Must be used in conjunction with .sqluser= property. Example: source[2].sqlserver = deltav9 source[2].sqldatabase = DVHisDB source[2].sqluser = Johns source[2].sqlpswd = test
	.isAE = true Optional for SQL data source Available in DeltaV 10.3+	Sets the SQL data source as the DeltaV Alarm and Events historian. Example: source[1].sqlserver = deltav10\DELTAV_CHRONICLE source[1].sqldatabase = Ejournal source[1].isae = true

Event Journals as Data Source

Event journals are files that are generated directly by a DeltaV Batch Execution System (BES). Each file represents execution of particular recipe and contains a log of batch events as well as batch related data. The interface expects that each record (row) in the event file will contain at least 15 tab-delimited columns which contain the following information in the following order:

Column1: Timestamp (either LclTime or GMTTime)
Column2: BatchID
Column3: Recipe
Column4: Descript
Column5: Event
Column6: Pvalue
Column7: EU
Column8: Area
Column9: ProcCell
Column10: Unit
Column11: Phase
Column12: PhaseDesc
Column13: UserID
Column14: UniqueID
Column15: Comment

SQL Batch Historian as Data Source

The DeltaV Batch Execution System (BES) consists of various components and the SQL Server is one of them. The SQL Server is used to store events performed by the DeltaV BES. While the SQL Server contains multiple databases used to store realtime data and batch data, the interface requires only primary for initial connection, by default named –**DVHisDB**.

Table 2. List of public views used for data retrieval.

View/Table	Description
Batchview	Contains UniqueID, BatchID, start time, end time, Product, UniqueID and archived flag with new archived database name for all batches.
Brecipestatechangeview	Contains State Change events which are used by default for PI batch generation.
Batchrecipeview	Contains Recipe data for all batches, such as Procedure, Unit Procedure, Operation, Phase, equipment linkage, start and end time for each object.
Batchequipmentview	Contains equipment arbitration for all batches.
Batcheventview (available when switch /uobev is specified in command line parameters)	Contains batch associated data for all batches. **Note:** This view does not provide explicit [Descript], [Pval] and [EU] fields. Instead the [Descript] field contains all 3 fields combined.
LocaleVars	This table is used to convert Local start and end times with DST offsets provided by each view to GMT time and then to UTC seconds.

View/Table	Description
Union of Tables: BactiveStepChangeEvent, BmaterialChargeRequestEvent BmaterialChargeEvent BpauseStatusEvent BequipmentSelectionEvent BphaseLinkPermissiveEvent BrecipeModeChangeEvent BrecipeModeCommandEvent BrecipeStateChangeEvent BrecipeStateCommandEvent BrecipeValueChangeEvent BrecipeValueRequestEvent BreportEvent BtextMessageEvent BphaseBatchRequestEvent BrecipeComment UnhandledBatchMsg (used by default to retrieve batch associated events)	Contains batch associated data for all batches. These tables are used by default to retrieve batch associated events. **Note:** the union of the tables provides the batch associated events with explicit fields [Descript], [Pval] and [EU].

If the data archiving was performed on DeltaV SQL batch historian, then the new database name containing archived batch will be referenced in the primary database – **DVHisDB**, allowing the interface to reconnect to the new database and retrieve archived batch data as well. Data retrieval from SQL server is scan based. Microsoft **ADO** driver for the Microsoft SQL is used to communicate with the SQL server databases (part of SQL Native client package). Microsoft SQL Native Client can be downloaded from the following location: http://msdn.microsoft.com/en-us/data/aa937733.aspx

SQL Alarm&Events Historian as Data Source

The DeltaV Batch Execution System (BES) 9.3+ includes the SQL Alarm&Events historian. This historian uses an independent SQL Server for its storage, which is named by default as <node name>\DELTAV_CHRONICLE. The stored data is identical to the data produced by the OPC DA Alarm&Events server. The default database name is **Ejournal**. The use of SQL A&E historian is optional and it is not required by the interface.

View/Table	Description
Journal	Each event contains the following fields: Timestamp, Event Type, Category, Area, Unit, Module, Module Description, Attribute, Level, Desc1, Desc2.

The SQL A&E data source(s) can be specified separately or in parallel with the SQL Batch historian data source(s). Note, the SQL AE data source is missing the ProcessCell definition. To properly map module hierarchy retrieved from SQL AE database onto existing DeltaV Batch equipment hierarchy, it is required to define the ProcessCell correlation to the Area or ProcessCell correlation to the Unit. It can be achieved through INI file definition:

Apcmap: <Area> = <ProcessCell>

And/Or

Upcmap: <Unit> = <ProcessCell>

where **apcmap** stands for **A**rea to **P**rocess**C**ell **m**ap. For DeltaV default PAINT recipe, the map can be defined as:

Apcmap: EXTERIOR_PAINT = PAINT_BLEND

UPCmap stands for **U**nit to **P**rocess**C**ell **map**. The mapping can be defined as:

Upcmap: U101 = PAINT_BLEND

Starting with version 3.0.0.x, the interface can use the DeltaV-generated equipment hierarchy XML file. Based on the DeltaV equipment structure provided , the interface searches the missing ProcessCell by matching the combination of Area and Unit. To enable this feature, specifythe following command line parameter:

/EquipmentXML=<file path>

Where <file path> is the path to DeltaV generated XML file.

Example: /EquipmentXML="c:\DeltaV\EquipHier.xml"

The definition of SQL AE historian as data source is identical to specifying the DeltaV batch SQL historian with one exception – an extra parameter **source[##].isAE=true** has to be defined.

Example 1: DeltaV Batch historian is the only data source:

```
Source[1].sqlserver = deltav10
Source[1].database = DVHisDB
```

Example 2: DeltaV Batch (1) and Alarm and Events (2) historians as data sources:

```
Source[1].sqlserver = deltav10
Source[1].database = DVHisDB
Source[2].sqlserver = deltav10\DELTAV_CHRONICLE
Source[2].database = Ejournal
Source[2].isAE = true
Apcmap: EXTERIOR_PAINT = PAINT_BLEND

Upcmap: U101 = PAINT_BLEND
```

Data retrieval from SQL server is scan based. Microsoft **ADO** driver for the Microsoft SQL is used to communicate with the SQL server databases (part of SQL Native client package). Microsoft SQL Native Client can be downloaded from the following location: http://msdn.microsoft.com/en-us/data/aa937733.aspx

OPC Alarm&Events Server as Data Source

OPCAE server is another component of the DeltaV Batch Execution System. Starting with version 10.3, OPCAE server provides batch events performed by the BES in real time. The use of OPCAE server is optional and is not required by the interface. Although, if it is used as an additional data source, it allows processing of batch data into the PI server in real time, not scan based. The OPCAE server can be specified only in conjunction with the DeltaV SQL server, which serves as backup source as well as the source for additional batch associated events.

Recipe Model vs. Equipment Model

Two distinct yet related models are used to describe batch processes. These are the Recipe Model and the Equipment Model. Diagrams depicting hierarchical structures, in particular those for the S88-compliant hierarchical structure, of these models are shown in **Figures 5** and **6**. The Equipment Model describes the physical equipment necessary to create a batch while the Recipe Model describes the procedures, which are performed during the execution of a recipe. There is **no** intrinsic or direct relationship between the levels of the Equipment

Model and the Recipe Model. With the exception of *Arbitration* events, journal files contain only Recipe Model event information.

It should be noted that within S88, the use of procedures and unit procedures is optional. A recipe may be defined consisting of only operations and phases.

Figure 5. Recipe Model hierarchy

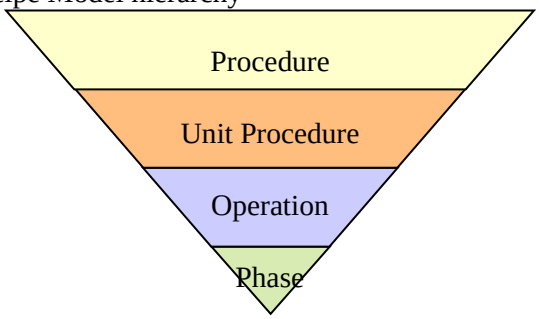
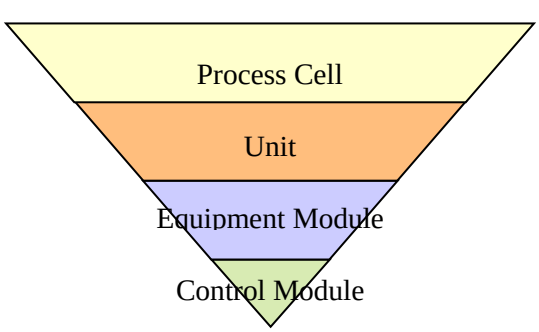


Figure 6. Equipment Model hierarchy



The Batch interface uses S88 terminology and hierarchy as framework to collate and store information in a structured manner within either PI Server (Module and Batch databases) or AF Server (elements and event frames).

When interface is configured to store batch data in PI Batch database, unit procedures are mapped to PI UnitBatches. This assumption implies that only a single unit procedure can be active in a unit at any given time. This lays a fundamental restriction on the configuration of recipes that may be run by the BES, if the Batch interface is to be used to process the resultant data and populate the BDB in a meaningful manner. If there are overlapping Unit Procedures on the same unit, the interface closes the conflicting PI UnitBatches, although the data is still processed into closed PI UnitBatches. The actual end time for truncated UnitBatch is stored in its Product property. The actual Product is appended by the keyword "_TrueEndUTC=" which is followed by UTC seconds representing the actual End Time for specific unit batch.

When the interface is configured to store batch data in AF server, there are no restrictions on how the batch data can be stored. AF event frames support parallel unit procedures natively.

If the recipe is divided in multiple smaller unit procedures or operations, the DeltaV Batch interface should be run with merge startup command line parameter for entering that data into the PI server. Please refer to the Merging Multiple Source batches into a Single PIBatch section for more information on how the merge works.

PI Batch Database Methodology

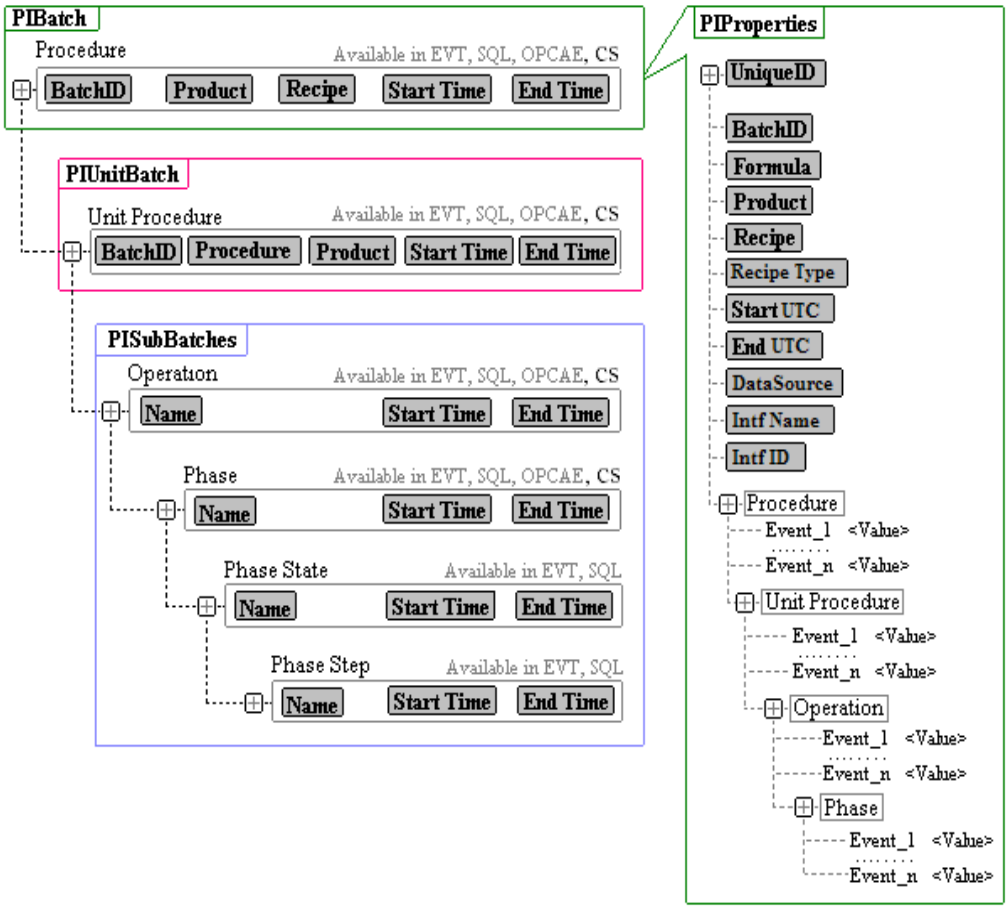
The PI Module and Batch Databases are used to organize and store batch data. Further discussion of these databases can be found in the PI 3.3 Data Archive Manual and the PI SDK tutorial documentation. This interface creates PIBatch, PIUnitBatch and hierarchy of PISubBatch objects within the PI Batch Database to represent the recipe procedures, unit procedures, operations, phases, phase states and phase steps respectively (Fig. 7). Each of the objects created in the PI Batch Database has the following common properties:

- Name (PISubBatch)
- Batch ID (PIBatch and PIUnitBatch objects only)
- Start time
- End time

Note that, in a PIBatch, the name is stored in the Recipe property and, in a PIUnitBatch the Procedure property is used to store the name of the corresponding recipe level. If illegal characters **(* ‘ ? | ` “)** are encountered in the BatchID, Name, Product, Recipe or Procedure fields, they are replaced with the underscore “_” character. Each object in the PI Batch Database represents a specific level of the Recipe Model. However, the relationship between the PI Batch Database and the Recipe Model is complicated by the possibility of building a recipe without the procedure or unit procedure levels. In cases where the highest recipe level is an operation or phase (i.e. neither procedure nor unit procedure levels are defined), PIBatch and PIUnitBatch objects must be still created by the interface.

The Batch interface can operate in two different processing modes, which are applicable only to EVT, SQL and OPCAE data sources. The first mode is the default mode of operation, which is based on State Change events. When the interface is run in this mode, you can change between data sources without PI Batch database time adjustments. The second mode of operation is Batch Recipe-based processing, enabled by specifying the **`/ubr`** parameter in command line parameters. This mode is only applicable to EVT or SQL data sources and changing between data source might create PI Batch database object time frame adjustments.

Figure 7. Schematic of PI Batch Database organization.



PIBatch

The PIBatch object is created for each batch defined in the data source. All records associated with the source batch can be recorded in the PIProperties collection of the PIBatch or in PI Points. The root PIProperty nodes are always the UniqueID of the batches which is assigned automatically by the Batch Executive. The interface stores the following batch properties under UniqueID: BatchID, Product, Formula Name, Recipe, Recipe Type, Start Time UTC, End Time UTC, Interface Name, Interface ID, DataSource, and events defined by the client. The underlying structure of the PIProperties collection is organized to reflect the hierarchy of the Recipe Model described by the data source where the Recipe names create hierarchical PIProperty nodes. Events of interest are stored in lists under appropriate Recipe node. Each PIProperty event name is defined as 'Event_<event count>', where <event count> is the current number of events already stored under a specific node. This method of event naming is dictated by the PIProperty rule, which states each event name under the same node should be unique. The PIProperty value can be defined through the use of Property templates. Please refer to Property Template section below for description and configuration steps.

The PIBatch represents the procedure within the recipe. Each PIBatch contains a collection of associated PI UnitBatches (which correspond to the Unit Procedures in the recipe).

The PIBatch object can represent merged object, which contains multiple source batches with identical BatchID or common subset of characters in BatchID. The PI Batch Product and Recipe properties contain data associated with the first source batch which started merged PI Batch. Use PIProperties to retrieve original source batch properties. For each merged source batch, the interface creates a node named as the UniqueID of the source batch containing the original batch properties.

Note: Because source batch can be terminated unexpectedly, without proper unloading by the operator, the interface maintains this batch in the local memory for 100 days, after which the batch is considered abandoned, and the interface closes the batch with the latest known time stamp for this particular batch. The abandon timeout can be changed through the command line parameter **`/abto=<days>`** (**A**bandoned **B**atch **T**ime**O**ut).

PI Batch Start Event Combinations

Data Source	PIBatch Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “CREATED” The batch recipe type is determined by the recipe messages such as system message – Procedure Started, Unit Procedure Started, etc.
DeltaV EVT with /ubr switch enabled	The batch recipe event containing: [Event] field = “System Message” and [Pvalue] field = “Beginning Of BATCH”. The associated [EU] field = “Procedure” / “Unit Procedure” / “Operation” / “Phase” determines the type of the particular recipe.
DeltaV SQL	The batch recipe event containing: [EventType] field = “State Change” and [EventDescript] field containing substring “CREATED” is used to set the Start Time for PIBatch. The batch recipe type is provided for each event by the data source in [Action] field. The event is retrieved from “dbo.brecipestatechangeview” or “dbo.batcheventview” views.
DeltaV SQL with /ubr switch enabled	The batch object with [ActivateTime] timestamp used to set the Start Time of PIBatch. This batch objects are retrieved from SQL “batchview” view. The batch recipe is determined by other batch recipe objects retrieved from “batchrecipeview” view.
DeltaV OPCAE	The BATCH-EVENT event with Event Attribute [6] = “LOAD” is used to start PIBatch object (0 based index) Note: batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc. Note: the PIBatch is not created until the interface is able to identify the recipe type of the batch, which is normally available as soon as the recipe is loaded and started.

PI Batch End-Triggering Event combinations

Data Source	PIBatch End-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “REMOVED”
DeltaV EVT with /ubr switch enabled	The first out of two recipe events is used to set an End Time for PIBatch object. 1. The batch recipe event containing [Event] field = “System Message” and [Pvalue] field = “End Of BATCH” 2. The batch recipe event containing: [Event] field = “State Change” and [Pvalue] field = “REMOVED”/ ”COMPLETE” / ”ABORTED”
DeltaV SQL	The batch recipe event containing: [EventType] field = “State Change” and [EventDescript] field containing substring “REMOVED” is used to set the End Time for PIBatch. This event is retrieved from “dbo.brecipestatechangeview” or “dbo.batcheventview” views.
DeltaV SQL with /ubr switch enabled	Uses source batch object [DeactivateTime] retrieved from SQL “dbo.batchview” table. Batch Recipe type is provided for each object by the data source.
DeltaV OPCAE	The BATCH-EVENT event with Event Attribute [6] = “REMOVED” is used to set an End Time for PIBatch object (0 based index) Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

PIUnitBatch

A PIUnitBatch is created for each unit procedure as defined in the data source. The start and end times of a PIUnitBatch are intended to reflect the onset and completion of physical processing within a unit.

The PIUnitBatch properties do not change if the parent object is a merged PI Batch. PIUnitBatch always contains original BatchID and Procedure name as it is defined in the source unless **`/tbid`** parameter was specified in command line parameters. This parameter enforces a stripped BatchID to be used for PIUnitBatch objects and for all events to be stored in PIPoints and PIProperties.

When Operation or Phase level recipes are run, the interface uses the Operation/Phase name as the PIUnitBatch Procedure name.

PIUnitBatch Start Time-Triggering Event Combinations

Data Source	PIUnitBatch Start-Triggering Event(s)
DeltaV EVT	For Procedure, Unit Procedure and Operation level recipes, the following two events must be preset to set the Start Time for PIUnitBatch. The timestamp of the latest event is used as the start time. • The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Phase Level recipes, single event is used to start PIUnitBatch containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure level recipes, the following two events must be preset to set the Start Time for PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Unit Procedure Started”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Operation level recipes the following two events must be present to start PIUnitBatch: • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit” with the [Pvalue] field containing the actual unit name. For Phase level recipes, single event is used to set the Start Time for PIUnitBatch containing [Event] field = “State Change”, [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	For Procedure, Unit Procedure and Operation level recipes, the following two events must be preset to start PIUnitBatch. The latest timestamps used as the start time. • The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” which is retrieved from “brecipestatechangeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase level recipes, single event is used to set the Start Time for PIUnitBatch, containing [EventType] field = “State Change” with [Pvalue] field containing substring “RUNNING”. The batch recipe hierarchy is provided in [Action] field.

Data Source	PIUnitBatch Start-Triggering Event(s)
DeltaV SQL with /ubr parameter enabled	For Procedure and Unit Procedure level recipes, the following two events must be preset to start PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing the [StartTime] timestamp associated with the specific “unitprocedure” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Operation level recipes, the following two events must be preset to start PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing the [StartTime] timestamp associated with the specific “operation” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase level recipes, the batch recipe event containing the [StartTime] associated with the specific “phase” object is sufficient to set the Start Time for PIUnitBatch. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase].
DeltaV OPCAE	The BATCH-EVENT event with the following attributes is used to set the Start Time for PIUnitBatch: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> RUNNING Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

PI UnitBatch End Time-Triggering Event Combinations

Data Source	PIUnitBatch End-Triggering Event(s)
DeltaV EVT	For Procedure, Unit Procedure and Operation level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE” / “STOPPED” / “ABORTED”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Phase Level recipes, single event is used to set an End Time for PIUnitBatch containing [Event] field = “State Change” and [Pvalue] field = ““COMPLETE” / “STOPPED” / “ABORTED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event containing [Event] column = “System Message” and [Descript] column = “Unit Procedure Finished”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Operation level recipes the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit” with the [Pvalue] field containing the actual unit name. For Phase level recipes, single event is used to set an End Time for the PIUnitBatch, containing [Event] field = “State Change” and [Pvalue] field = “COMPLETED” / “ABORTED” / “STOPPED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	For Procedure, Unit Procedure and Operation level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event: [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETED” / “ABORTED” / “STOPPED” which is retrieved from “brecipestatechangeview” view. • The arbitration event containing [ReleaseTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase level recipes, single event is used to set an End Time for PIUnitBatch, containing [EventType] field = “State Change” with [Pvalue] field containing substrings “COMPLETED” / “ABORTED” / “STOPPED”. The batch recipe hierarchy is provided in [Action] column.

Data Source	PIUnitBatch End-Triggering Event(s)
DeltaV SQL with /ubr parameter enabled	For Procedure and Unit Procedure level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event containing the [EndTime] associated with the specific “unitprocedure” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [ReleaseTime] associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Operation level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: • The batch recipe event containing the [EndTime] associated with the specific “operation” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [ReleaseTime] associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase level recipes, the batch recipe event containing the [EndTime] associated with the specific “phase” object is sufficient to set an End Time for PIUnitBatch. The batch recipe hierarchy is provided in columns: [Procedure], [UnitProcedure], [Operation] and [Phase].
DeltaV OPCAE	The BATCH-EVENT with the following attributes is used to set an End Time for PIUnitBatch: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETED / ABORTED / STOPPED **Note**: Batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

PISubBatches

Operation

A PISubBatch is created for each source operation found within the data source as child for PIUnitBatch object.

Note: The operation and phase level recipes populate upper levels of PIBatch Database hierarchy automatically with PIUnitBatch Procedure property and PISubBatch operation name as the name of the source Operation/Phase recipe object.

PISubBatch Operation Start-Triggering Events

Data Source	PISubBatch Operation Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes, the batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started” is used to set the Start Time for PISubBatch operation level object. For Phase level recipes the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING” is used to set the Start Time for PISubBatch operation level object. The [Recipe] field contains the batch recipe hierarchy.

Data Source	PISubBatch Operation Start-Triggering Event(s)
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the Start Time for PISubBatch operation level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes the batch recipe event containing the [StartTime] timestamp associated with the specific source “operation” object is used to set the Start Time for PISubBatch operation level object. The event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase level recipes the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the Start Time for PISubBatch operation level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV OPCAE	For Procedure, Unit Procedure, Operation, Phase level recipes, the BATCH-EVENT event with the following attributes is used to set the Start Time for PISubBatch operation level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> RUNNING Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

PISubBatch Operation End triggering events

Data Source	PISubBatch Operation End triggering event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE” / “ABORTED” / “STOPPED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes, the first event out of two following events is used to set an End Time for PISubBatch operation level object: the batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished” The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “REMOVED” (at Operation level). Note, this event is used due to possibility that some “Operation Finished” events are not present in EVT data source. For Phase level recipes the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING” is used to set the Start Time for PISubBatch operation level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETE” / “ABORTED” / “STOPPED” is used to set an End Time for PISubBatch operation level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.

Data Source	PISubBatch Operation End triggering event(s)
DeltaV SQL with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes the batch recipe event containing the [EndTime] timestamp associated with the specific source “operation” object is used to set an End Time for PISubBatch operation level object. This event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase level recipes the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETE” / “ABORTED” / “STOPPED” is used to set an End Time for PISubBatch operation level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV OPCAE	For Procedure, Unit Procedure, Operation, Phase level recipes, the BATCH-EVENT event with the following attributes is used to set the Start Time for PISubBatch operation level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETE / ABORTED / STOPPED Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Phase

A PISubBatch is created for each phase found within the data source as child for Operation level PISubBatch object.

Note: The phase level recipes populate upper levels of PIBatch Database hierarchy automatically with PIUnitBatch Procedure property and PISubBatch operation name as the name of the source Phase recipe object.

PISubBatch Phase Start-Triggering Events

Data Source	PISubBatch Phase Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure, Operation, Phase level recipes, the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING” is used to set the Start Time for PISubBatch phase level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the Start Time for PISubBatch phase level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.

Data Source	PISubBatch Phase Start-Triggering Event(s)
DeltaV SQL with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes the batch recipe event containing the [StartTime] timestamp associated with the specific source “phase” object is used to set the Start Time for PISubBatch phase level object. The event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase level recipes the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the Start Time for PISubBatch phase level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV OPCAE	For Procedure, Unit Procedure, Operation, Phase level recipes, the BATCH-EVENT event with the following attributes is used to set the Start Time for PISubBatch phase level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> RUNNING Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

PISubBatch Phase End-Triggering Events

Data Source	PISubBatch Phase End-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE”/”STOPPED”/”ABORTED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure, Unit Procedure, Operation, Phase level recipes, the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE” / ”STOPPED” / ”ABORTED” is used to set an End Time for PISubBatch phase level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “COMPLETE” / ”STOPPED” / ”ABORTED” is used to set an End Time for PISubBatch phase level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure, Unit Procedure, Operation level recipes the batch recipe event containing the [EndTime] timestamp associated with the specific source “phase” object is used to set an End Time for PISubBatch phase level object. The event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase level recipes the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “COMPLETE” / ”STOPPED” / ”ABORTED” is used to set an End Time for PISubBatch phase level object. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.

Data Source	PISubBatch Phase End-Triggering Event(s)
DeltaV OPCAE	For Procedure, Unit Procedure, Operation, Phase level recipes, the BATCH-EVENT event with the following attributes is used to set an End Time for PISubBatch phase level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETE / ABORTED / STOPPED Note, batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Phase State

A PISubBatch is created for each phase state found within the data source as child for Phase level PISubBatch object. All Phase States are sequential; start of new Phase State ends the previous Phase State. Note, the self terminating Phase States which set its End Times are COMPLETE, ABORTED and STOPPED.

PISubBatch Phase State-Triggering Events

Data Source	PISubBatch Phase State-Triggering Event
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = <State Name>. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = <State Name>. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescr] field containing substring <State Name>. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	The batch recipe event containing [EventType] field = “State Change” and [EventDescr] field containing substring <State Name>. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV OPCAE	No Phase State data available from this source.

Phase Step

A PISubBatch is created for each phase step found within the data source as a child for the Phase State level PISubBatch object. Phase Steps are not S88-complaint and are custom to each particular implementation and configuration of the Batch Execution System. By default this level of PISubBatches is not enabled. To enable this feature use the optional switch **`/ras=<Start Substring>, <End Substring>`** (**R**eport **A**s **S**tep). The Phase Steps are always created beneath the first PISubBatch Phase State = “RUNNING”, regardless if the parent Phase State is ended or not. The Phase Step name and start/stop events are coming from the “Descript” column. The triggering event is “Report”. The Phase Steps do not create the higher level PI Batches, UnitBatches and SubBatches, if the parent Phase is not found. If the Phase Step was not closed by the appropriate closing event, it will be closed by the end of the parent Operation level PI SubBatch. 0-duration Phase Steps are ignored. Multiple sequential Start/End events are ignored except the first one.

PISubBatch Phase Step Start-Triggering Events

Data Source	PISubBatch Phase State Start-Triggering Event
DeltaV EVT	The following two events can set the Start Time for PISubBatch phase step object. The event containing [Event] field = "Report" and [Descript] field containing **<Start Substring>**. The Phase Step name is determined as the prefix substring to <Start Substring> in [Descript] field. The event containing [Event] field = "Report" and [Pvalue] field containing **<Start Substring>**. The Phase Step name is determined as the prefix substring to <Start Substring> in [Pvalue] field. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The event containing [EventType] field = "Report" and [EventDescr] field containing **<Start Substring>** is used to set the Start Time for PISubBatch Phase State object. The Phase Step name is determined as the prefix substring to <Start Substring> in [EventDescr] field. The associated [Action] field contains the batch recipe hierarchy
DeltaV OPCAE	No Phase Step data available from this source.

PISubBatch Phase Step End-Triggering Events

Data Source	PISubBatch Phase State Start-Triggering Event
DeltaV EVT	The following two events can set an End Time for PISubBatch phase step object. The event containing [Event] field = "Report" and [Descript] field containing **<End Substring>**. The Phase Step name is determined as the prefix substring to <End Substring> in [Descript] field. The event containing [Event] field = "Report" and [Pvalue] field containing **<End Substring>**. The Phase Step name is determined as the prefix substring to <End Substring> in [Pvalue] field. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The event containing [EventType] field = "Report" and [EventDescr] field containing **<End Substring>** is used to set an End Time for PISubBatch Phase State object. The Phase Step name is determined as the prefix substring to <End Substring> in [EventDescr] field. The associated [Action] field contains the batch recipe hierarchy
DeltaV OPCAE	No Phase Step data available from this source.

Arbitration Events Unavailable

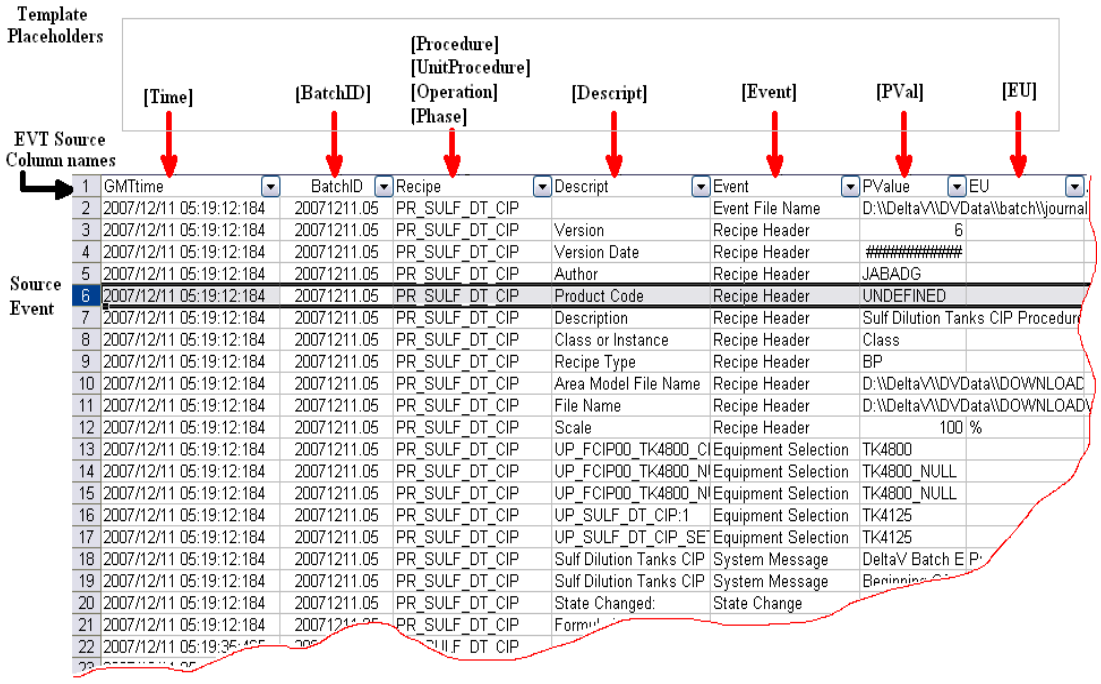
The behavior described above is the default behavior of the interface. However, if the batch execution system does not generate arbitration events, you can select the option "**Disable Arbitration**" (`/noarbitration`). With this option, the start time of PIUnitBatches is determined by the later of either "Unit Procedure Started" or the start of a sublevel (Operation or Phase) event in that unit. The end time of PIUnitBatches is determined by the earlier of the "Unit Procedure Finished" message and end of the last sublevel in that unit. If no unit procedures have been defined (that is, operation is the highest recipe level), the start of the first phase marks the start time of the PIUnitBatch, Operation level PISubBatch and the first Phase level PISubBatch. The end of the last phase (as determined by the presence of the "Operation Finished" system message) marks the end time of the PIUnitBatch, Operation PISubBatch, and that Phase PISubBatch. In this case, the PIUnitBatch and PISubBatch have the same start and end times.

Note: If arbitration events are unavailable, the triggering of the PIUnitBatches from the Event File is only imprecisely bracketed by the Unit Procedure Started/Finished events and the start/end of the sublevels of the recipe. The arbitration event is the most explicit method of determining the allocation of a physical unit to a given instance of a recipe.

Template Placeholders

The PI Interface to the Emerson DeltaV Batch uses templates to specify what is stored in PI Batch Product field, PI Batch Properties, PI Points and it also allows defining the equipment hierarchy structure in the PI Module Database. The word – template is used to define the custom name and/or value structure applied to particular PI object. The template is defined by a combination of a free text and placeholders. The placeholder can be referred as to the name of the column in EVT or SQL data sources, except the Recipe column which is broken down into subcolumns, such as Procedure, UnitProcedure, Operation and Phase. Placeholders are used to identify the positions in the name or value structure where the actual data from these columns is going to be placed. Figure 8 provides the sample placeholder association to the EVT column names.

Figure 8. Placeholders to EVT columns association example.



	GMTtime	BatchID	Recipe	Descript	Event	PValue	EU
2	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP		Event File Name	D:\\DeltaV\\DVData\\batch\\journal	
3	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version	Recipe Header	6	
4	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version Date	Recipe Header	############	
5	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Author	Recipe Header	JABADG	
6	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Product Code	Recipe Header	UNDEFINED	
7	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Description	Recipe Header	Sulf Dilution Tanks CIP Procedur	
8	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Class or Instance	Recipe Header	Class	
9	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Recipe Type	Recipe Header	BP	
10	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Area Model File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOAD	
11	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOADV	
12	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Scale	Recipe Header	100	%
13	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_CI	Equipment Selection	TK4800	
14	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
15	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
16	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP:1	Equipment Selection	TK4125	
17	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP_SE	Equipment Selection	TK4125	
18	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	DeltaV Batch E P	
19	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	Beginning C	
20	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	State Changed:	State Change		
21	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Formu			
22	2007/12/11 05:19:36:4		ULF_DT_CIP				

Example:

Consider the following template structure:

```
Sample [Time] | [Descript]-[BatchID]:[Event]__pvalue:[Pval][EU]
```

This structure contains the free text and the placeholder combination. Assume that the incoming event is the row number 6 (Figure 8), which is **Recipe Header**. Then using the template structure we can replace placeholders with the actual data from the associated columns to create the following text:

```
Sample 2007/12/11 05:19:12:184 | Product Code:Recipe Header__pvalue:UNDEFINED
```

Note, in this example [EU] placeholder was replaced with BLANK value since the source row did not have the associated column populated.

PIBatch and PIUnitBatch Product Property

Some of data source contain the Product ID information. In the case of Event Files, this information is stored as the Pvalue in the row that contains the description "Product Code". Typically this is a Recipe Header event.

However, if there is some other description instead of "Product Code", then the language translation section of the INI file can be used to change the description value that the interface looks for. For example, if the "Product Value" is used in the EVT file, then the language translation can be set so that "Product Value" is the same as "Product Code". The language translation can be achieved through the use of INI file. The language translation syntax is the following and it is not case sensitive:

```
translate: <Foreign language definition> = <English definition>,
```

Example:

```
Translate: Product Value = Product Code
```

Note, the translations approach is only applicable to the EVT data sources. SQL data sources contain the Product field as part of the source batch object and cannot be changed with language translations.

Another way to change the PIBatch and PIUnitBatch Product property is through the use of Product Template, which can be defined in the interface's INI file. This template represents the order in which the interface tries to set the PIBatch product field when the batch start event received. The product template: product=<list of product options>. It is possible to use placeholders such as [product] or [formula] in conjunction with free text to set the product.

Product Template Definition

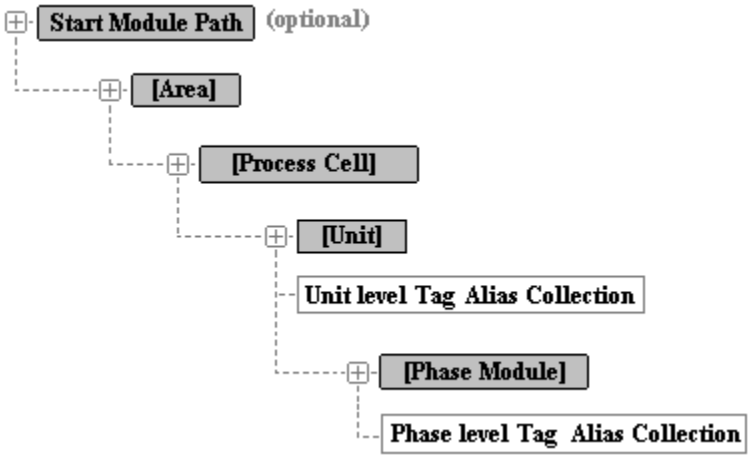
Template Name	Value	Allowed Placeholders	Description
product	<list of options>	[product] [formula]	Defines the PI Batch and PI UnitBatch product name. The provided value should be a list of possible product sources. From most important to least important. Allowed placeholders are not case sensitive. Default: Product=[product] Example 1: Product=[Product],[Formula],TestProduct In this example, when batch starts, it checks if the "Source Product" is defined, if yes, it uses the actual product name, if product is missing, the interface checks if the "Source Formula Name" is known, if yes, then it uses the Formula name as product. In the worst case scenario, when there is no product or formula defined before start of batch, the value "TestProduct" is used as product name. Example 2: Product=abs:[Formula], def_[Product], Unknown On start time of source batch, assume, that Source Formula=12345aaa Source Product=IceCream Then resulting PI Batch Product=abs:12345aaa Assume that Formula is not known, then PI Batch Product=def_IceCream Assume that both Product and Formula are unknown, then PI Batch Product = Unknown

In case of a merged PIBatch, the PIBatch Product contains the product associated with the batch that started the PIBatch. All merged batch products are stored in PIProperties underneath UniqueID nodes.

PIModule Creation

The interface automatically creates modules and units in the PI Server. PI Units (PIModules with the IsUnit flag set to TRUE) are created if needed when they are first encountered in the data source. The following modules in DeltaV equipment hierarchy are maintained by the interface: Area, Process Cell, Unit and Phase Module. By default, the placement of these modules is at the root level of the Module DB. Although, the user can define an optional Starting Module Path by using the (**`/smp`** command line parameter) under which the equipment hierarchy will be created. The default structure of the PI module hierarchy utilized by the interface is depicted in Figure 9.

Figure 9. Interface PI Module DB Structure



The DeltaV Batch automatically references PI tags at the unit and phase PIModules through tag aliases if tag names contain unit and phase module names. If the tag name contains reference to the unit and not the phasemodule, then only Unit Alias is created for this tag.

If for any reason default equipment hierarchy is not feasible, then the interface allows creating custom equipment hierarchy through the use of Equipment Template (Table n). Equipment Template should be defined in INI file associated with specific interface instance.

Note, if SQL Alarm&Events SQL server is defined as one of the data source(s), the interface will use this Equipment template definition only down to [Unit] level. All underlying module names will be named exactly as it is provided by the data source.

PI AF Event Frames Methodology

PI Event Frames are used to organize and store the batch data. The event frame hierarchy supports the storage of source batch data with no data manipulations. At any level of the hierarchy, each event frame has its own set of attributes, which provide the ability to store source batch attributes under a specific event frame regardless of its depth. This approach is a major improvement over the PI Batch database, where only the top level object (PIBatch) can contain attributes (PIProperties). To bypass this limitation, artificial PIProperty nodes that represent the hierarchical structure of the source recipe were created in PIBatch properties or the data was stored in PI Tags.

Each event frame has the following fields:

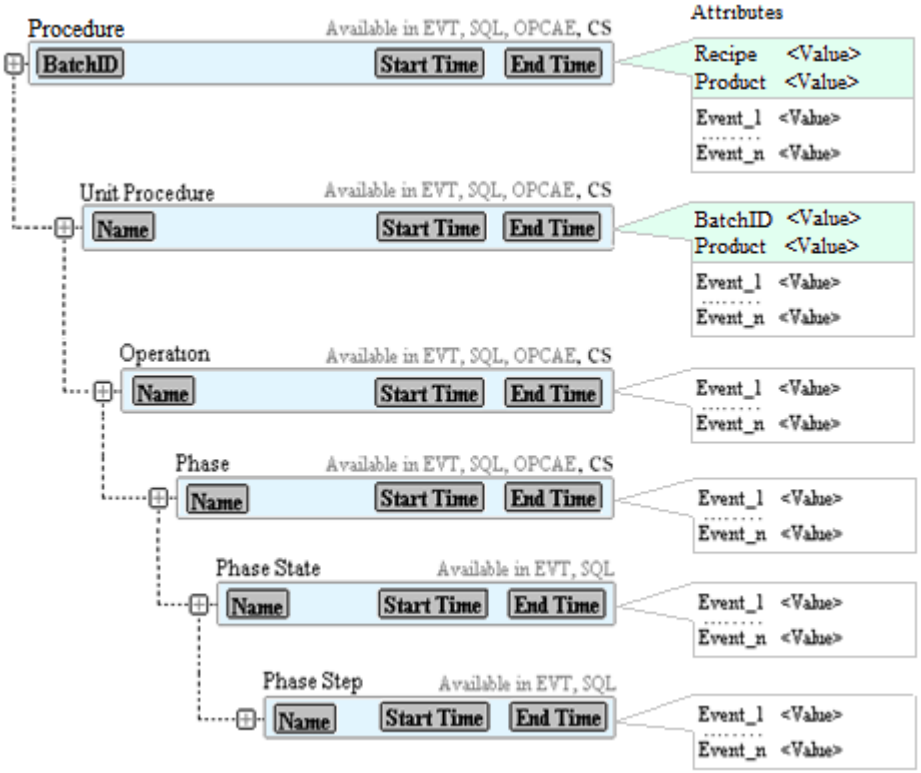
- Name
- Description
- Start Time
- End Time
- Template
- Category
- Event Specific Attributes
- Referenced elements (such as Unit, Phase Module)

In the root level event frame, the Name field contains the source batch BatchID. For sub event frames, the Name field contains the actual recipe name, such as UnitProcedure, Operation, Phase, etc. To maintain compatibility with the way data is stored in PI Batch Database, the following source batch properties are stored as event frame attributes. For Procedure-level recipe, source batch Product and Recipe properties are stored as event frame attributes. For UnitProcedure-level recipes, the source batch BatchID and Product are stored as event frame attributes.

If illegal characters **(* ‘ ? | ` “)** are encountered in the Name field, they are replaced with the underscore “_” character. Each object in the event frame represents a specific level of the Recipe Model. However, the relationship between the event frames and the Recipe Model is complicated by the possibility of building a recipe without the procedure or unit procedure levels. In cases where the highest recipe level is an operation or phase (that is, neither procedure nor unit procedure levels are defined), event frames that correspond to Procedure and UnitProcedure level must be still created by the interface.

The Batch interface can operate in two different processing modes, which are applicable only to EVT, SQL and OPCAE datasources. The first mode is default mode of operation and based on State Change events. When the interface is run in this mode, you can change between data sources without PI Batch database time adjustments. The second mode is Batch Recipe-based processing, which can be enabled by specifying the **`/ubr`** parameter in the command line. This mode is only applicable to EVT or SQL data sources, and changing between data source can create PI Batch database object time frame adjustments.

Figure 7b. Schematic of AF event frames organization.



Procedure

The root event frame is created for each batch defined in the data source, and represents the Procedure in the Recipe. Each root event frame contains a collection of associated child event frames that correspond to the Unit Procedures in the recipe. All records associated with the source batch can be recorded in the Attributes collection of the event frame or in PI Points. Because source batches can have identical BatchID and Recipe Names within the same timeframe, the interface stores additional information in the Extended Properties of the root event frame, to match the source batch with an event frame. The Extended Properties are the flat Name-Value collection. The Name is the BatchID of the source batch, which is automatically assigned by the Batch Executive, and the value is the XML structure containing the following batch properties: BatchID, Product, Formula Name, Recipe, Recipe Type, Start Time UTC, End Time UTC, Interface Name, Interface ID, DataSource.

To maintain maintain compatibility with PI Batch database, the root event frame Name is the BatchID of the the source batch. The "Recipe" (Procedure Name) and "Product" properties are stored as the searchable attributes of the event frame. Below is the source batch property mapping to an AF event frame.

Source Procedure Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced Elements
BatchID	Name		
Procedure Name		Recipe	
Product		Product	
Start Time	Start Time		
End Time	End Time		
	Template="Procedure"	default Attributes: Recipe, Product	
	Category="OSIBatch"		

In addition to "Product" and "Recipe" Attributes, by default, the interface captures the following batch-associated events and stores them in Procedure-level AF event frame Attributes:

Souce Batch "Event" Column Values	AF Attribute Name	AF Attribute Value	AF Attribute UOM	Category
Recipe Header	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Recipe Header
Formula Header	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Formula Header
Recipe Value	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Recipe Value
Report	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Report

The Procedure-level event frame can represent a merged object, which contains multiple source batches with identical BatchIDs or a common subset of characters in its BatchID. The Product and Recipe attributes contain data associated with the first source batch that started

the merged event frame. For each merged source batch, the interface creates a node in Extended Properties of the event frame, named with the UniqueID of the source batch and the value containg the XML containing the original source batch properties.

Note: A source batch can terminate unexpectedly without proper unloading by the operator. The interface maintains such a batch in the local memory for 100 days, after which the batch is considered abandoned and the interface closes the batch with the latest known time stamp for this particular batch. The abandon timeout can be changed using the command line parameter **`/abto=<days>`** (**A**bandoned **B**atch **T**ime**O**ut).

Procedure-Level AF Event Frame Start Event Combinations

Data Source	AF Event Frame Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “CREATED” The batch recipe type is determined by the recipe messages, such as the system message “Procedure Started, Unit Procedure Started”, etc.
DeltaV EVT with /ubr switch enabled	The batch recipe event containing: [Event] field = “System Message” and [Pvalue] field = “Beginning Of BATCH”. The associated [EU] field = “Procedure” / “Unit Procedure” / “Operation” / “Phase” determines the type of the particular recipe.
DeltaV SQL	The batch recipe event containing: [EventType] field = “State Change” and [EventDescript] field containing substring “CREATED” is used to set the Start Time for PIBatch. The batch recipe type is provided for each event by the data source in the [Action] field. The event is retrieved from “dbo.brecipestatechangeview” or “dbo.batcheventview” views.
DeltaV SQL with /ubr switch enabled	The batch object with [ActivateTime] timestamp used to set the Start Time of PIBatch. This batch objects are retrieved from SQL “batchview” view. The batch recipe is determined by other batch recipe objects retrieved from “batchrecipeview” view.
DeltaV OPCAE	The BATCH-EVENT event with Event Attribute [6] = “LOAD” is used to start PIBatch object (0 based index) **Note**: The batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc. The Procedure-level event frame is not created until the interface is able to identify the recipe type of the batch, which is normally available as soon as the recipe is loaded and started.

Procedure-Level AF Event Frame End-Triggering Event Combinations

Data Source	AF Event Frame End-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “REMOVED”
DeltaV EVT with /ubr switch enabled	The first out of two recipe events is used to set an End Time for PIBatch object. • The batch recipe event containing [Event] field = “System Message” and [Pvalue] field = “End Of BATCH” • The batch recipe event containing: [Event] field = “State Change” and [Pvalue] field = “REMOVED”/ ”COMPLETE” / ”ABORTED”
DeltaV SQL	The batch recipe event containing: [EventType] field = “State Change” and [EventDescript] field containing substring “REMOVED” is used to set the End Time for PIBatch. This event is retrieved from “dbo.brecipestatechangeview” or “dbo.batcheventview” views.

Data Source	AF Event Frame End-Triggering Event(s)
DeltaV SQL with /ubr switch enabled	Uses source batch object [DeactivateTime] retrieved from SQL "dbo.batchview" table. Batch Recipe type is specified for each object by the data source.
DeltaV OPCAE	The BATCH-EVENT event with Event Attribute [6] = "REMOVED" is used to set an End Time for PIBatch object (0 based index) The batch recipe type is determined by any of the BATCH-EVENT's containing explicit actions, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

UnitProcedure

A UnitProcedure-level event frame is created for each unit procedure as defined in the data source. Each UnitProcedure-level event frame is created as a child of the Procedure-level event frame and contains the subset of event frames that represent the source batch Operation-level recipe.The start and end times of an event frame are intended to reflect the onset and completion of physical processing in a unit. The parallel UnitProcedures are supported completely by AF event frames. That is, the event frame End Time reflects the actual end time of the source UnitProcedure,

The name field of the UnitProcedure-level event frames reflects an actual source batch UnitProcedure name. To maintain compatibility with PI Batch database, the interface stores the "BatchID" and the "Product" source batch properties as searchable attributes of the event frame. Below is the source unit procedure mapped to eventframe fields and attributes:

Source UnitProcedure Properties	AF Event Frame Fields	AF Event Frame Attributes	Referenced Elements
BatchID		BatchID	
UnitProcedure Name	Name	Procedure	
Product		Product	
Start Time	Start Time		
End Time	End Time		
Unit			Unit
	Template="UnitProcedure"	default Attributes: BatchID, Procedure, Product	
	Category="OSIBatch"		

In addition to "BatchID, "Procedure" and "Product" Attributes, the interface captures and stores the following batch-associated events as event frame attributes:

Souce Batch "Event" Column Values	AF Attrubute Name	AF Attribute Value	AF Attribute UOM	Category
Recipe Value	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Recipe Value
Report	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Report

The UnitProcedure-level event frame properties do not change if the parent object is a merged event frame. UnitProcedure event frames always contains the original BatchID and Procedure name as defined in the source, unless the **`/tbid`** parameter was specified in the command line. This parameter configures a stripped BatchID to be used for UnitProcedure event frames objects and for all events to be stored in PIPoints and event frame attributes.

When Operation or Phase-level recipes are run, the interface uses the Operation/Phase name as the UnitProcedure-level event frame name.

UnitProcedure AF EventFrame Start Time-Triggering Event Combinations

Data Source	AF Event Frame Start-Triggering Event(s)
DeltaV EVT	For Procedure-, Unit Procedure- and Operation-level recipes, the following two events must be preset to set the Start Time for the PIUnitBatch. The timestamp of the latest event is used as the start time. • The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Phase-level recipes, a single event is used to start the PIUnitBatch, containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure- and Unit Procedure-level recipes, the following two events must be preset to set the Start Time for the PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Unit Procedure Started”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Operation-level recipes, the following two events must be present to start a PIUnitBatch: • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Acquired by recipe” and [EU] field = “Unit” with the [Pvalue] field containing the actual unit name. For Phase-level recipes, a single event is used to set the Start Time for PIUnitBatch, containing [Event] field = “State Change”, [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.

Data Source	AF Event Frame Start-Triggering Event(s)
DeltaV SQL	For Procedure-, Unit Procedure- and Operation-level recipes, the following two events must be preset to start a PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” which is retrieved from “brecipestatechangeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase-level recipes, single event is used to set the Start Time for PIUnitBatch, containing [EventType] field = “State Change” with [Pvalue] field containing substring “RUNNING”. The batch recipe hierarchy is provided in [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure- and Unit Procedure-level recipes, the following two events must be preset to start a PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing the [StartTime] timestamp associated with the specific “unitprocedure” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Operation-level recipes, the following two events must be preset to start PIUnitBatch. The latest timestamp is used as the start time. • The batch recipe event containing the [StartTime] timestamp associated with the specific “operation” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [AcquireTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase-level recipes, the batch recipe event containing the [StartTime] associated with the specific “phase” object is sufficient to set the Start Time for PIUnitBatch. The batch recipe hierarchy is provided in fields: [Procedure], [UnitProcedure], [Operation] and [Phase].
DeltaV OPCAE	The BATCH-EVENT event with the following attributes is used to set the Start Time for PIUnitBatch: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> RUNNING **Note:**,The batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

UnitProcedure AF EventFrame End Time-Triggering Event Combinations

Data Source	AF EventFrame End Triggering Event(s)
DeltaV EVT	For Procedure-, Unit Procedure- and Operation-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE” / “STOPPED” / “ABORTED”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Phase-Level recipes, a single event is used to set the end time for a PIUnitBatch, containing [Event] field = “State Change” and [Pvalue] field = ““COMPLETE” / “STOPPED” / “ABORTED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure- and Unit Procedure-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event containing [Event] column = “System Message” and [Descript] column = “Unit Procedure Finished”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit”. The [Pvalue] field contains the actual unit name. For Operation-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished”. • The arbitration event containing [Event] field = “Recipe Arbitration”, [Descript] field = “Resource Released by recipe” and [EU] field = “Unit” with the [Pvalue] field containing the actual unit name. For Phase-level recipes, a single event is used to set the end time for a PIUnitBatch, containing [Event] field = “State Change” and [Pvalue] field = “COMPLETED” / “ABORTED” / “STOPPED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	For Procedure-, Unit Procedure- and Operation-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event: [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETED” / “ABORTED” / “STOPPED” which is retrieved from “brecipestatechangeview” view. • The arbitration event containing [ReleaseTime] timestamp associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase-level recipes, a single event is used to set the end time for a PIUnitBatch, containing [EventType] field = “State Change” with [Pvalue] field containing substrings “COMPLETED” / “ABORTED” / “STOPPED”. The batch recipe hierarchy is specified in the [Action] column.

Data Source	AF EventFrame End Triggering Event(s)
DeltaV SQL with /ubr parameter enabled	For Procedure- and Unit Procedure-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event containing the [EndTime] associated with the specific “unitprocedure” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [ReleaseTime] associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Operation-level recipes, the first of the following two events is used to set the end time for a PIUnitBatch: • The batch recipe event containing the [EndTime] associated with the specific “operation” object retrieved from the “batchrecipeview” view. • The arbitration event containing the [ReleaseTime] associated with the specific unit arbitration object retrieved from “batchequipmentview” view. For Phase-level recipes, the batch recipe event containing the [EndTime] associated with the specific “phase” object is sufficient to set the end time for a PIUnitBatch. The batch recipe hierarchy is specified in the following columns: [Procedure], [UnitProcedure], [Operation] and [Phase].
DeltaV OPCAE	The BATCH-EVENT with the following attributes is used to set the end time for a PIUnitBatch: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETED / ABORTED / STOPPED **Note**, The batch recipe type is determined by any of the BATCH-EVENT’s containing an explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Operation

An Operation-level event frame is created for each Operation as defined in the data source. Each Operation-level event frame is created as a child of the UnitProcedure-level event frame, and contains the subset of event frames that represent the source batch Phase-level recipe.

The Name field of the Operation-level event frames reflects an actual source recipe Operation name. Below is the source operation recipe mapping to event frame fields and attributes:

Source Operation Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced elements
Operation Name	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
	Template=”Operation”		
	Category=”OSIBatch”		

By default, the interface captures and stores the following batch associated events as event frame attributes:

Souce Batch “Event” Column Values	AF Attrubute Name	AF Attribute Value	AF Attribute UOM	Category

Recipe Value	Source “Description” Column value	Source “Pvalue” column value	Source “EU” column value	Recipe Value
Report	Source “Description” Column value	Source “Pvalue” column value	Source “EU” column value	Report

Note: The operation- and phase-level recipes populate upper levels of the event frames hierarchy automatically from the Procedure-level event frame attribute: “Recipe” and UnitProcedure/Operation-level event frame Name field as the name of the source Operation/Phase recipe object.

Operation-Level AF Event Frame Start-Triggering Events

Data Source	AF Event Frame Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started” is used to set the start time for PISubBatch operation level object. For Phase-level recipes, the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING” is used to set the start time for PISubBatch operation level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the start time for PISubBatch operation-level object. The event is retrieved from the “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is specified in event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the batch recipe event containing the [StartTime] timestamp associated with the specific source “operation” object is used to set the start time for a PISubBatch operation-level object. The event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is specified in the following fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase-level recipes, the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substring “RUNNING” is used to set the start time for a PISubBatch operation-level object. The event is retrieved from the “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is specified in event’s [Action] field.
DeltaV OPCAE	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the BATCH-EVENT event with the following attributes is used to set the start time for PISubBatch operation level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> RUNNING **Note**, The batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Operation-Level AF Event Frame End-Triggering Events

Data Source	AF Event Frame End-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “COMPLETE” / “ABORTED” / “STOPPED”. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the first event of two following events is used to set the end time for a PISubBatch operation-level object: The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished” The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “REMOVED” (at Operation level). This event is used due to the possibility that some “Operation Finished” events are not present in EVT data source. For Phase-level recipes, the batch recipe event containing [Event] field = “State Change” and [Pvalue] field = “RUNNING” is used to set the start time for PISubBatch operation-level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETE” / “ABORTED” / “STOPPED” is used to set the end time for a PISubBatch operation-level object. The event is retrieved from the “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the batch recipe event containing the [EndTime] timestamp associated with the specific source “operation” object is used to set the end time for a PISubBatch operation-level object. This event is retrieved from the “batchrecipeview” view. The batch recipe hierarchy is provided in the following fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase-level recipes, the batch recipe event containing [EventType] field = “State Change” and [EventDescript] field containing substrings “COMPLETE” / “ABORTED” / “STOPPED” is used to set the end time for PISubBatch operation-level object. The event is retrieved from the “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in the event’s [Action] field.
DeltaV OPCAE	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the BATCH-EVENT event with the following attributes is used to set the start time for a PISubBatch operation-level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETE / ABORTED / STOPPED **Note:** The batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Phase

A Phase-level event frame is created for each Phase, as defined in the data source. Each Phase-level event frame is created as a child of the Operation-level event frame and contains the subset of event frames that represent the source batch Phase States-level recipe.

The name field of the Phase-level event frame reflects an actual source recipe Phase name. Below is the source phase recipe mapping to event frame fields and attributes:

Source Phase Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced elements
Phase Name	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
Phase Module			Phase Module
	Template="Phase"		
	Category="OSIBatch"		

By default, the interface captures and stores the following batch associated events as event frame attributes:

Souce Batch "Event" Column Values	AF Attrubute Name	AF Attribute Value	AF Attribute UOM	Category
Recipe Value	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Recipe Value
Report	Source "Description" Column value	Source "Pvalue" column value	Source "EU" column value	Report

Note: The Phase-level recipes automatically populate upper levels of the event frames hierarchy with the Procedure-level event frame Recipe attribute UnitProcedure- and Operation-level event frame Name field as the name of the source Phase recipe object.

Phase-Level AF Event Frame Start-Triggering Events

Data Source	AF Event Frame Start-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = "State Change" and [Pvalue] field = "RUNNING". The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the batch recipe event containing [Event] field = "State Change" and [Pvalue] field = "RUNNING" is used to set the start time for a PISubBatch phase- level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = "State Change" and [EventDescript] field containing substring "RUNNING" is used to set the start time for a PISubBatch phase-level object. The event is retrieved from the "brecipestatechangeview" / "batcheventview" view.

Data Source	AF Event Frame Start-Triggering Event(s)
	The batch recipe hierarchy is specified in the event's [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the batch recipe event containing the [StartTime] timestamp associated with the specific source "phase" object is used to set the start time for a PISubBatch phase-level object. The event is retrieved from the "batchrecipeview" view. The batch recipe hierarchy is provided in the following fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase-level recipes, the batch recipe event containing [EventType] field = "State Change" and [EventDescript] field containing substring "RUNNING" is used to set the start time for a PISubBatch phase-level object. The event is retrieved from the "brecipestatechangeview" / "batcheventview" view. The batch recipe hierarchy is specified in the event's [Action] field.
DeltaV OPCAE	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the BATCH-EVENT event with the following attributes is used to set the start time for a PISubBatch phase-level object: Event Attribute [6] = "State Changed" Event Attribute [8] = <batch recipe hierarchy> RUNNING **Note:** The batch recipe type is determined by any of the BATCH-EVENT's containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Phase-Level AF Event Frame End-Triggering Events

Data Source	AF Event Frame End-Triggering Event(s)
DeltaV EVT	The batch recipe event containing [Event] field = "State Change" and [Pvalue] field = "COMPLETE"/"STOPPED"/"ABORTED". The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the batch recipe event containing [Event] field = "State Change" and [Pvalue] field = "COMPLETE" / "STOPPED" / "ABORTED" is used to set the end time for a PISubBatch phase-level object. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = "State Change" and [EventDescript] field containing substring "COMPLETE" / "STOPPED" / "ABORTED" is used to set the end time for a PISubBatch phase-level object. The event is retrieved from the "brecipestatechangeview" / "batcheventview" view. The batch recipe hierarchy is provided in the event's [Action] field.
DeltaV SQL with /ubr parameter enabled	For Procedure-, Unit Procedure- and Operation-level recipes, the batch recipe event containing the [EndTime] timestamp associated with the specific source "phase" object is used to set the end time for a PISubBatch phase-level object. The event is retrieved from the "batchrecipeview" view. The batch recipe hierarchy is provided in the following fields: [Procedure], [UnitProcedure], [Operation] and [Phase]. For Phase-level recipes, the batch recipe event containing [EventType] field = "State Change" and [EventDescript] field containing substring "COMPLETE" / "STOPPED" / "ABORTED" is used to set theend time for a PISubBatch phase-level object. The event is retrieved from the "brecipestatechangeview" / "batcheventview" view. The batch recipe hierarchy is specified in the event's [Action] field.

Data Source	AF Event Frame End-Triggering Event(s)
DeltaV OPCAE	For Procedure-, Unit Procedure-, Operation- and Phase-level recipes, the BATCH-EVENT event with the following attributes is used to set the end time for a PISubBatch phase-level object: Event Attribute [6] = “State Changed” Event Attribute [8] = <batch recipe hierarchy> COMPLETE / ABORTED / STOPPED **Note**, The batch recipe type is determined by any of the BATCH-EVENT’s containing explicit action, such as Procedure Started/Finished, UnitProcedure Started/Finished, etc.

Phase State

A Phase State-level event frame is created for each Phase State, as defined in the data source. Each Phase State-level event frame is created as a child of the Phase-level event frame and, if configured, can contain the subset of event frames that represent the Phase Step objects. All Phase States are sequential; the start of new Phase State ends the previous Phase State.

The self terminating Phase States that set its end times are COMPLETE, ABORTED and STOPPED. These Phase States have a zero-duration timeframe.

The name field of the Phase State event frames reflects an actual source recipe Phase State name. Below is the mapping of source phase state to event frame fields and attributes:

Source Phase State Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced Elements
Phase State	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
Phase Module			Phase Module
	Template=”Phase State”		
	Category=”OSIBatch”		

Phase State-Level AF Event Frame-Triggering Eevents

Data Source	AF Event Frame-Triggering Event
DeltaV EVT	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = <State Name>. The [Recipe] field contains the batch recipe hierarchy.
DeltaV EVT with /ubr parameter enabled	The batch recipe event containing [Event] field = “State Change” and [Pvalue] field = <State Name>. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The batch recipe event containing [EventType] field = “State Change” and [EventDescr] field containing substring <State Name>. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in the event’s [Action] field.
DeltaV SQL with /ubr parameter enabled	The batch recipe event containing [EventType] field = “State Change” and [EventDescr] field containing substring <State Name>. The event is retrieved from “brecipestatechangeview” / ”batcheventview” view. The batch recipe hierarchy is provided in event’s [Action] field.
DeltaV OPCAE	No Phase State data is available from this source.

Phase Step

An event frame is created for each Phase Step found in the data source as a child of the Phase State-level event frame. Phase Steps are not S88-compliant and are unique to each particular implementation and configuration of the Batch Execution System. By default, this level of event frames is not enabled. To enable this feature, specify the optional switch **`/ras=<Start Substring>, <End Substring>`** (**R**eport **A**s **S**tep). Phase Steps are always created beneath the first Phase State EventFrame Name = "RUNNING", regardless of whether the parent Phase State is ended. The Phase Step name and start/stop events come from the "Descript" column. The triggering event is "Report". If the parent Phase is not found, Phase Steps do not create the higher-level Procedure-, UnitProcedure-, Operation- or Phase-level event frames. If the Phase Step was not closed by the appropriate closing event, it is closed by the end of the parent Operation-level event frame. Zero-duration Phase Steps are ignored. Multiple sequential Start/End events are ignored, except the first one.

Below is the source phase step mapped to event frame fields and attributes:

Phase Step Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced elements
Unit			Unit
Phase Module			Phase Module
	Template="Phase Step"		
	Category="OSIBatch"		

Phase Step AF Event Frame Start-Triggering Events

Data Source	AF Event Frame Start-Triggering Event
DeltaV EVT	The following two events can set the Start Time for a PISubBatch phase step object. The event containing [Event] field = "Report" and [Descript] field containing **<Start Substring>**. The Phase Step name is determined from the prefix substring to <Start Substring> in [Descript] field. The event containing [Event] field = "Report" and [Pvalue] field containing **<Start Substring>**. The Phase Step name is determined from the prefix substring to <Start Substring> in [Pvalue] field. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The event containing [EventType] field = "Report" and [EventDescr] field containing **<Start Substring>** is used to set the start time for a PISubBatch Phase State object. The Phase Step name is determined from the prefix substring to <Start Substring> in [EventDescr] field. The associated [Action] field contains the batch recipe hierarchy
DeltaV OPCAE	No Phase Step data is available from this source.

Phase Step AF Event Frame End-Triggering Events

Data Source	AF Event Frame Stop-Triggering Event
DeltaV EVT	The following two events can set the end time for a PISubBatch phase step object. The event containing [Event] field = "Report" and [Descript] field containing **<End Substring>**. The Phase Step name is determined from the prefix substring to <End Substring> in [Descript] field. The event containing [Event] field = "Report" and [Pvalue] field containing **<End Substring>**. The Phase Step name is determined from the prefix

Data Source	AF Event Frame Stop-Triggering Event
	substring to <End Substring> in [Pvalue] field. The [Recipe] field contains the batch recipe hierarchy.
DeltaV SQL	The event containing [EventType] field = "Report" and [EventDescr] field containing **<End Substring>** is used to set the end time for a PISubBatch Phase State object. The Phase Step name is determined from the prefix substring to <End Substring> in [EventDescr] field. The associated [Action] field contains the batch recipe hierarchy
DeltaV OPCAE	No Phase Step data is available from this source.

Arbitration Events Unavailable

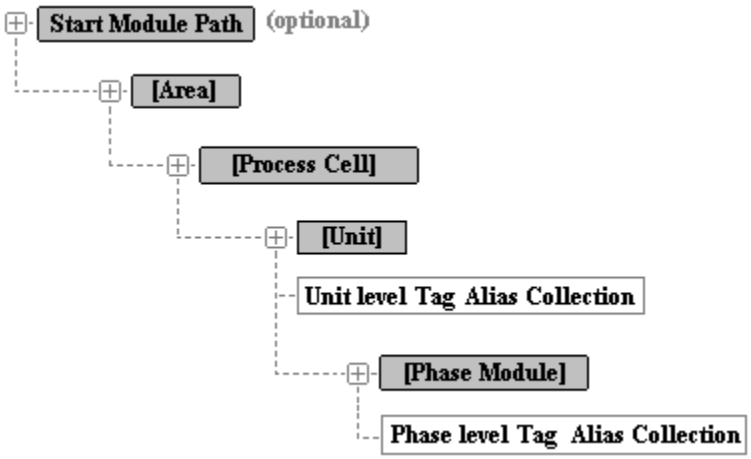
The behavior described above is the default behavior of the interface. However, if the batch execution system does not generate arbitration events, can may select the option "**Disable Arbitration**" (**`/noarbitration`**). With arbitration disabled, the start time of UnitProcedure EventFrame is determined by the later of either "Unit Procedure Started" or the start of a sublevel (Operation or Phase) event in that unit. The end time of a UnitProcedure event frame is determined by the earlier of the "Unit Procedure Finished" message and the end of the last sublevel in that unit. If no unit procedures have been defined (that is, Operation is the highest recipe level), the start of the first phase marks the start times of the UnitProcedure, Operation and the first Phase event frames. The end of the last phase (as determined by the presence of the "Operation Finished" system message) marks the end time of the UnitProcedure, Operation and that Phase event frames. In this case, the UnitProcedure and Operation event frames have the same start and end times.

Note: If arbitration events are unavailable, the triggering of the PIUnitBatches from the event file is imprecisely bracketed by the Unit Procedure Started/Finished events and the start/end of the sublevels of the recipe. The arbitration event is the most precise method of detecting the allocation of a physical unit to a specific instance of a recipe.

PI AF Element Creation

The interface creates the PI AF elements that represent the source equipment hierarchy. PI AF elements are created if needed when they are first encountered in the data source. The following elements in the DeltaV equipment hierarchy are maintained by the interface: Area, Process Cell, Unit and Phase Module. By default, these elements reside at the root level of the element collection. You can define an optional starting element path under which the equipment hierarchy will be created by specifying the **`/smp`** command line parameter . The default structure of the PI AF elements hierarchy used by the interface is depicted in Figure 9a.

Figure 9a. Interface PI AF Element Structure



The DeltaV Batch automatically references PI tags for the unit and phase modules PI AF elements through tag aliases if tag names contain unit and phase module names. If the tag name contains reference to the unit and not the phase module, only the Unit Alias for this tag is created.

If the default equipment hierarchy is not feasible for any reason, the interface supports creating a custom equipment hierarchy using an equipment template. The equipment template must be specified in the INI file associated with the specific interface instance.

If the SQL Alarm&Events SQL server is defined as a data source, the interface uses the equipment template definition only to the [Unit] level. All underlying PI AF elements are named as defined by the data source.

Foreign Language Support

The PI Interface to the Emerson DeltaV Batch supports languages other than English by providing the use of a look-up table for the various event types that trigger specific actions in the interface. Note that this is not necessarily all of the events that are possible in the data source, only a selected few are required: those, which are used to trigger the start and end times of the PI Batch objects (Table 5). The language translation syntax is the following:

translate: <native language definition> = <foreign language definition>,

where translation is identified by the keyword – **translate:** followed by the native language word or phrase to be translated. The equal sign determines the end of the native language phrase or word definition and serves as a token for the start of English definition. The syntax is not case sensitive.

DELTAV EVT (default)

DeltaV EVT required English definitions (no `/ubr` switch):

Category	Can be found in EVT Column	English Definitions
S88 Levels		
	[EU]	Procedure
	[EU]	Unit Procedure
	[EU]	Operation
	[EU]	Phase
Batch Header info		
	[Event]	Recipe Header
	[Descript]	Product Code
	[Event]	Formula Header
	[Descript]	Formula Name
Equipment Arbitration		
	[Event]	Recipe Arbitration
	[Descript]	Resource Acquired by recipe
	[Descript]	Resource Released by recipe
	[EU]	Unit
Batch Recipe		
	[Event]	System Message
	[Pval]	Beginning Of BATCH
	[Pval]	End Of BATCH
	[Event]	State Change
State Changes		
	[Pval]	CREATED
	[Pval]	RUNNING
	[Pval]	COMPLETE
	[Pval]	REMOVED
	[Pval]	STOPPED
	[Pval]	ABRORTED
Additional		
	[Event]	Comment
(if /ras switch is used)	[Event]	Report

DELTAV EVT (with /ubr switch defined)

DeltaV EVT required English definitions (with `/ubr` switch)

Category	Can be found in EVT Column	English Definitions
S88 Levels		
	[EU]	Procedure
	[EU]	Unit Procedure
	[EU]	Operation

Category	Can be found in EVT Column	English Definitions
	[EU]	Phase
Batch Header info		
	[Event]	Recipe Header
	[Descript]	Product Code
	[Event]	Formula Header
	[Descript]	Formula Name
Equipment Arbitration		
	[Event]	Recipe Arbitration
	[Descript]	Resource Acquired by recipe
	[Descript]	Resource Released by recipe
	[EU]	Unit
Batch Recipe		
	[Event]	System Message
	[Pval]	Beginning Of BATCH
	[Pval]	End Of BATCH
	[Descript]	Unit Procedure Started
	[Descript]	Unit Procedure Finished
	[Descript]	Operation Started
	[Descript]	Operation Finished
	[Event]	State Change
State Changes		
	[Pval]	RUNNING
	[Pval]	COMPLETE
	[Pval]	REMOVED
	[Pval]	STOPPED
	[Pval]	ABRORTED
Additional		
	[Event]	Comment
(if /ras switch is used)	[Event]	Report

DELTAV SQL (default: no /ubr switch)

Table of DeltaV SQL required English definitions:

Category	Can be found in SQL Column	English Definitions
Batch Recipe		
	[EventType]	State Change
State Changes		
	[EventDescr]	CREATED
	[EventDescr]	RUNNING
	[EventDescr]	COMPLETE
	[EventDescr]	REMOVED

Category	Can be found in SQL Column	English Definitions
	[EventDescr]	STOPPED
	[EventDescr]	ABRORTED
Additional		
(if /ras switch is used)	[EventType]	Report

DELTAV SQL (with /ubr switch)

DeltaV SQL required English definitions (with `/ubr` switch):

Category	Can be found in SQL Column	English Definitions
Batch Recipe		
	[EventType]	State Change
State Changes		
	[EventDescr]	RUNNING
	[EventDescr]	COMPLETE
	[EventDescr]	REMOVED
	[EventDescr]	STOPPED
	[EventDescr]	ABRORTED
Additional		
(if /ras switch is used)	[EventType]	Report

DELTAV SQL +OPCAE (NO /ubr switch available)

DeltaV SQL +OPCAE required English definitions:

Category	Can be found in SQL Column	OPCAE BATCH-EVENT attribute index (0-based)	English Definitions
Equipment Arbitration			
	N/A	Attr[6]	Resource Acquired by recipe
	N/A	Attr[6]	Resource Released by recipe
Batch Recipe			
	N/A	Attr[6]	Procedure Started
	N/A	Attr[6]	Procedure Finished
	N/A	Attr[6]	Unit Procedure Started
	N/A	Attr[6]	Unit Procedure Finished
	N/A	Attr[6]	Operation Started
	N/A	Attr[6]	Operation Finished
	N/A	Attr[6]	Phase Started
	N/A	Attr[6]	Phase Finished
	[EventType]	N/A	State Change
	N/A	Attr[6]	State Changed
State Changes			
	N/A	Attr[6]	LOAD

Category	Can be found in SQL Column	OPCAE BATCH-EVENT attribute index (0-based)	English Definitions
	[EventDescr]	N/A	CREATED
	[EventDescr]	Attr[8]	RUNNING
	[EventDescr]	Attr[8]	COMPLETE
	[EventDescr]	Attr[6], Attr[8]	REMOVED
	[EventDescr]	Attr[8]	STOPPED
	[EventDescr]	Attr[8]	ABRORTED
Additional			
(if /ras switch is used)	[EventType]	N/A	Report

Sample German EVT required and optional event translations (when `/ubr` is used):

```
//  [S88 Levels]
 translate: "Grundrezept"        = "Procedure"
 translate: "Teilrezept"         = "Unit Procedure"
 translate: "Grundoperation"     = "Operation"
 translate: "Grundfunktion"      = "Phase"

// [Batch Header info]
translate: "Rezeptkopf"          = "Recipe Header"
translate: "Produktcode"         = "Product Code"
translate: "Formelkopf"          = "Formula Header"
translate: "Formelname"          = "Formula Name"

// [Arbitrations]
translate: "Rezeptzuteilung"     = "Recipe Arbitration"
translate: "Betriebsmittel belegt durch Rezept" = "Resource Acquired
by recipe"
translate: "Betriebsmittel freigegeben durch Rezept" = "Resource
Released by recipe"
translate: "Teilanlage"          = "Unit"

// [Recipe Logic, Comment needed only if there is an extra column in
DeltaV, so event
// can be converted to Comment event, and use English word in Tag
Template as
// trigger]

translate: "Zustands\E4\nderung" = "State Change"
translate: "Kommentar"           = "Comment"

translate: "Systemmeldung"             = "System Message"
translate: "CHARGEN-Anfang"            = "Beginning Of BATCH"
translate: "CHARGEN-Ende"              = "End Of BATCH"
translate: "Teilrezept gestartet"      = "Unit Procedure Started"
translate: "Teilrezept beendet"        = "Unit Procedure Finished"
translate: "Grundoperation gestartet"  = "Operation Started"
translate: "Grundoperation beendet"    = "Operation Finished"

// [Phase States]
translate: "L\C4\UFT"            = "RUNNING"
translate: "BEENDET"             = "COMPLETE"
```

```
translate: "ENTFERNT"            = "REMOVED"
translate: "GESTOPPT"            =  STOPPED"
translate: "ABGEBROCHEN"         = "ABORTED"

// [Optional assuming /ras switch is used]
translate: "Bericht"             = "Report"
```

Note: As soon as we translate word/phrase into English, we can use English Names, i.e. word – "Report" in resulting output of Template Tags and Properties. The templates must be defined using native language. Translations can be used to redefine any source word/phrase to any new word/phrase.

Template Example

Based on provided translations the interface can create and populate PI Tags and PI Properties from templates in native language. For example, consider the following tag template:

```
Tag[1].Name = [Unit] abc_[Parameter,value="Bericht"]
Tag[1].Value = [Value]:def
Tag[1].type = string
Tag[1].unitalias = Some Bericht
Tag[1].Descriptor = Bericht for Unit: [Unit]
Tag[1].EngUnits = just text

Property[1].Value = [Time] [Parameter,value="Bericht"] [Unit]-
[Value]
```

These templates are triggered by parameter – **Bericht** (German word for – Report) as it is found in the data source and the tag name and PI Property value will be based on native language parameter name. Assume that there is a particular row with unit field containing value: **"U101"** and associated Value field containing value: **"testing"**. In this case PI Tag name is defined as: **"U101 abc_Bericht"** and tag value is defined as: **"testing:def"**.

With the use of language translations we can create tag where foreign word/phrase is replaced with translated word/phase. For example:

```
translate:  "Bericht" = "Report"
translate:  "testing"  = "1"
```

The resulting tag name is **"U101 abc_Report"** and tag value is **"1:def"**

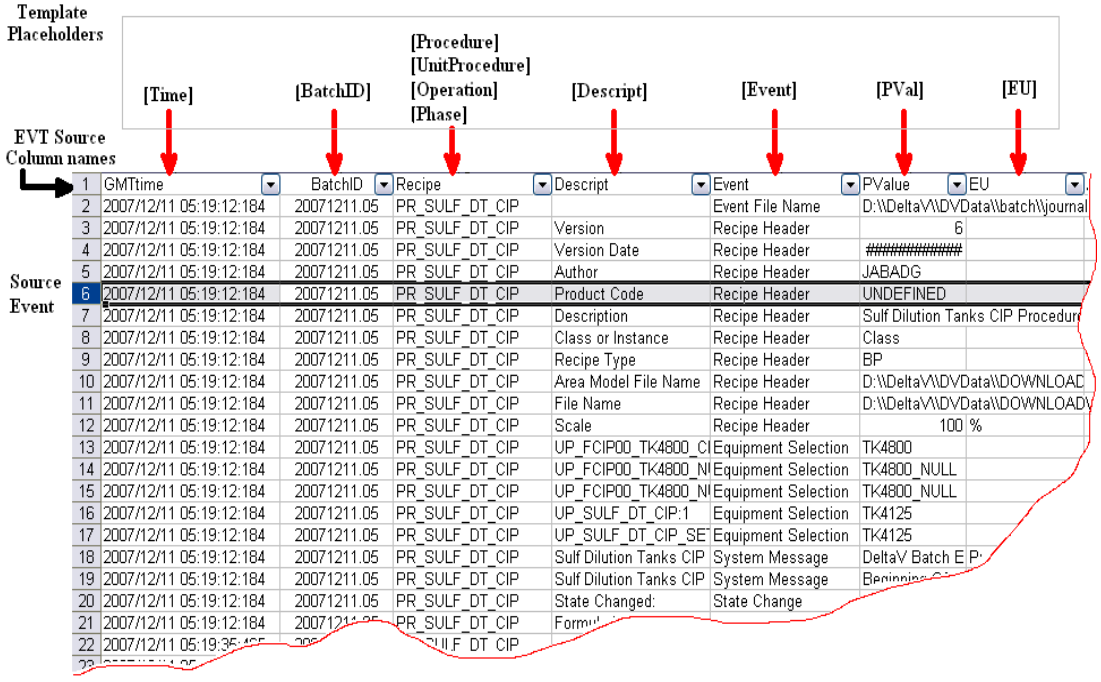
The same logic is applicable to the property template definitions. Translations are not case sensitive. Language translations don't have to be defined prior to the tag or property template definitions, they can be defined anywhere within the INI file.

Template Placeholders

The PI Interface to the Emerson DeltaV Batch uses templates to specify what is stored in the PI Batch Database, PI Module Database, AF event frames, AF elements and PI points. Templates define the custom name and/or value structure applied to a particular PI object. The template is defined using a combination of a free text and placeholders. The placeholder can be referred as to the name of the column in EVT or SQL data sources, except the Recipe column, which is composed of subcolumns, such as Procedure, UnitProcedure, Operation and Phase. Placeholders are used to identify the positions in the name or value structure where the

actual data from these columns is going to be placed. Figure 8 provides the sample placeholder association to the EVT column names.

Figure 8. Placeholders to EVT columns association example.



	GMTtime	BatchID	Recipe	Descript	Event	PValue	EU
2	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP		Event File Name	D:\\DeltaV\\DVData\\batch\\journal	
3	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version	Recipe Header	6	
4	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version Date	Recipe Header	###########	
5	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Author	Recipe Header	JABADG	
6	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Product Code	Recipe Header	UNDEFINED	
7	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Description	Recipe Header	Sulf Dilution Tanks CIP Procedur	
8	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Class or Instance	Recipe Header	Class	
9	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Recipe Type	Recipe Header	BP	
10	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Area Model File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOAD	
11	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOADV	
12	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Scale	Recipe Header	100	%
13	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_CI	Equipment Selection	TK4800	
14	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
15	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
16	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP:1	Equipment Selection	TK4125	
17	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP_SE	Equipment Selection	TK4125	
18	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	DeltaV Batch E	P
19	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	Beginning S	
20	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	State Changed:	State Change		
21	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Formu			
22	2007/12/11 05:19:35:4		ULF_DT_CIP				

For example, consider the following template structure:

```
Sample [Time] | [Descript]-[BatchID]:[Event]__pvalue:[Pval][EU]
```

This structure contains free text and placeholders. Assume that the incoming event is row 6 (in Figure 8), which is **Recipe Header**. Using the template structure, the placeholders are replacedwith the actual data from the associated columns to create the following text:

```
Sample 2007/12/11 05:19:12:184 | Product Code:Recipe Header__pvalue:UNDEFINED
```

In this example, the [EU] placeholder was replaced with a blank value, because the associated column in the source row was not populated.

Event Logging

Besides the creation and population of PI Batch, Module DB objects, AF event frames and AF elements, there are three methods by which the specific events from the data sources can be saved into the PI system. These methods are:

- The interface can store the individual events to the PI Properties hierarchy of the PI Batch or AF EventFrame Attributes when incoming event(s) match triggering events defined in Property Templates.
- The interface can create new PI Tags (and link them to a unit and phase module with a PI Aliases) when the incoming event(s) match triggering events defined in Tag Templates.
- The interface can create new Alarm PI Tags (and link them to a modules with a PI Aliases) when the incoming event(s) match triggering events defined in AlarmTag Templates.

These functions are separate actions and triggered independently under different sets of criteria.

DeltaV Batch Historian available placeholders:

```
[TIME], [BATCHID], [PROCEDURE], [UNITPROCEDURE], [OPERATION],
[PHASE], [DESCRIPT], [EVENT] or [PARAMETER], [PVAL] or [VALUE],
[EU], [AREA], [PROCESSCELL], [UNIT], [PHASEMODULE], [USERID] or
[USER], [UNIQUEID], [COMMENT]
```

DeltaV Alarm&Events Historian available placeholders:

```
[TIME], [EVENT], [CATEGORY], [NODE], [AREA], [PROCESSCELL], [UNIT],
[MODULE], [MODULEDESC], [ATTRIBUTE], [STATE], [LEVEL], [DESC1],
[DESC2]
```

For DeltaV Alarm&Events Historian, if the Module Description is “Unit Module”, then the `[UNIT]` placeholder contains the module name, else it contains the unit name.

Starting with version 1.0.1.0 interface supports wildcards in templates. Available wildcards are given below and can be used in any property field of Tag or Property Templates.

Wildcard	Description
#	single digit numerical value (0-9)
@	single alpha character (a-z, A-Z)
?	any single valid symbol
*	An array of valid symbols
!	repeat previous mask symbol

Example 1:

```
Tag[1].Name = [Event, value=”Report”]
```

or using mask

```
Tag[1].Name = [Event, value=”Repo*”]
```

Starting with version 2.0.1.x interface supports != (Not Equal) logic for the placeholder values. This logic is designed to be used inside the Triggering expressions. It allows ignoring particular event values from triggering the templates such as Tag, AlarmTag, Property and Recipe.

Example 2:

```
Recipe[1].Product=[Pval]

Recipe[1].ProductTrigger=[Event,value="Recipe Header"]
[Descript,value="Product Code"] [Pval, value!="UNDEFINED"]

Recipe[1].ProductTrigger=[Event,value="Formula Header"]
[Descript,value="Formula Name"]
```

In this example, the interface is going to ignore first triggering expression if the PVal field for the Product Code event contains the "UNDEFINED" string as a value. Because the Product Code event is the only one per batch, the second trigger takes the priority and when the triggering condition is met the PIBatch Product property is populated with the Formula Name value.

Advanced Parsing Parameters

Starting with interface version 1.0.1.0, each placeholder can contain additional parameters which allow advanced parsing of incoming data. The allowed syntax:

[Placeholder, <list of parameters delimited by comma>**]**

The following table provides the list of available parameters which can be used with any placeholder. Note, the names of parameters, placeholders, and value substrings are not case sensitive. If additional parameters are used for at least one placeholder, then in case of resulting substring returning empty set, the whole template will be set to blank. If it is desired to search in ALL fields of incoming event then set **Name of Placeholder** as a wild card, i.e. [*,lbe=”u:”].

Parameter	Description
VALUE=”substring” or “mask”	Defines the exact field value to search for in particular column. Masks are allowed. If ‘*’ is used instead of Name of Placeholder (i.e. search in all event fields, then [*,value=”test”] is equivalent to <test>
LBE=”substring” Optional	**L**eft **B**ound **E**xclusive substring. Defines the left bound of the target substring value. The resulting substring DOES NOT include the LBE defined boundary substring.
LBI=”substring” Optional	**L**eft **B**ound **I**nclusive substring. Defines the left bound of the target substring value. The resulting substring includes the LBI defined boundary substring.
RBE=”substring” Optional	Right Bound Exclusive substring. Defines the right bound of the target substring value. The resulting substring DOES NOT include the RBE defined boundary substring.
RBI=”substring” Optional	Right Bound Inclusive substring. Defines the right bound of the target substring value. The resulting substring includes the RBI defined boundary substring.
Delim=”substring” Optional	Delimiter character or substring. Must be used in conjunction with the Count parameter. This parameter defines the field separator. If used, it narrows the resulting substring to the substring contained within delimiters, where the starting delimiter index is specified by the count parameter. Note: right and left boundary substrings can be specified as well, resulting in parsing the delimited substring.
Count=# Optional	Index of the delimiter from which to start parsing. Must be used in conjunction with the Delim parameter.

For example, assume that [Value] column field contains the following data:

|U:browntod|C:SP_CHARGE_AMOUNT|O:1200|N:1123|E:kg|M:Local

The sample placeholder parameter combinations and results are presented in the table below.

Placeholder syntax	Resulting substring
[value]	|U:browntod|C:SP_CHARGE_AMOUNT|O:1200|N:1123|E:kg|M:Local
[value, lbe=”N:”]	1123|E:kg|M:Local
[value, lbi=”N:”]	N:1123|E:kg|M:Local
[value, rbe=”tod”]	|U:brown
[value, rbi=”tod”]	|U:browntod
[value, lbe=”U:”, rbe=”|”]	Browntod

Placeholder syntax	Resulting substring
[value, lbi="U:", rbe="|"]	U:browntod
[value, lbe="O:", rbi="kg"]	1200|N:1123|E:kg
[value, delim="|",count=3]	O:1200
[value, delim="|",count=3,lbe="O:"]	1200
[value, delim="|",count=3,lbe="C:SP",rbe="UNT"]	_CHARGE_AMO
[value, delim="|",count=6,lbe="M:"]	Local

Attribute / Property Templates

An interface supports both PI Batch and AF event frames for storing batch data. When the interface is configured to use AF Server as its host, batch data is stored as event frames and batch-associated data is stored in AF attributes. AF attibutes are part of each event frame that enable batch-data to be stored with the DeltaV execution event. All levels of the event frame hierarchy can store batch data in AF attributes.

When the interface is configured to run only against the PI Server, batch-recipe-associated data can be stored at the PIBatch level (root level) of the recipe hierarchy only by using the PIProperties collection, due to PI Server limitations. To maintain the recipe hierarchy, PIProperties are organized as a recipe tree, where each PIProperty node is the name of the recipe level, (procedure, unit procedure, operation, or phase). The data is stored in (Name, Value) lists under each node.

Note: The batch PI Properties collection has a limitation of 1Mb per PIBatch object. Therefore, it is not recommended to store all incoming events into batch PIProperties collection.

By default the interface does not store batch associated data into PIProperties. To store data in PIProperties, use Property Templates which define the subset of events and associated PIProperty value structure for each event to be stored in PIProperties. The Property Templates are not case sensitive and must be defined in the INI file associated with each specific instance of the interface. The Property Template can define only PIProperty values, but not the PIProperty names. This is dictated by the PISDK rule that states each PIProperty event name under the same PIProperty node should be unique. Each PIProperty name can be defined explicitly or if not defined, interface assigns the name as – 'Event_<event count>', where <event count> is the current number of events already stored under specific PI Property node. To configure the templates, use the Attribute[index] or Property[index] keywords as follows:

```
   Property[index].Name = free text without placeholders (hierarchy
   supported) (optional)
   Property[index].Value = free text without placeholders
   Property[index].Trigger = free text without placeholders
   Property[index].Translate = true/false (default: false)
   Property[index].EngUnits = free text without placeholders (AF
only)
   Property[index].Type = integer/float/string/auto
   Property[index].Category = free text without placeholders (AF
   only)
```

Or with keyword as "Attribute":

```
   Attribute[index].Name = free text without placeholders (hierarchy
   supported) (optional)
   Attribute[index].Value = free text without placeholders
   Attribute[index].Trigger = free text without placeholders
   Attribute[index].Translate = true/false (default: false)
   Attribute[index].EngUnits = free text without placeholders (AF
only)
   Attribute[index].Type = integer/float/string/auto
   Attribute[index].Category = free text without placeholders (AF
only)
```

where index is an arbitrary 1-based positive number. Specify the value structure in the free text format with the placeholder names contained within the square brackets. The triggering expression must be embedded in the Value Structure or specified through explicit Trigger(s). Specifying multiple placeholders in a single triggering expression is treated as an AND logic and specifying multiple trigger expressions is treated as an OR logic.

In cases when the source Engineering Units (UOM) do not match the AF Server Engineering Units (UOM), it is possible to define the conversion map through UOMMAP keyword. There can be multiple maps defined.

Syntax:

UOMMAP: <Source UOM> = <AF UOM>

Example:

UOMMAP: \\B0\\C = DEGC

To write Properties under the UniqueID PIProperty node, regardless of the source recipe sublevel from which the triggering event originated, add $ symbol as the first element in name path as shown below:

```
    Property[1].Name = $\[Parameter]
Or
   Attribute[1].Name = $\[Parameter]
```

To write Properties under PIBatch root PIProperties, regardless of the source recipe sublevel from which the triggering event originated, add @ symbol as the first element in name path as shown below:

```
    Property[1].Name = @\[Parameter]
Or
   Attribute[1].Name = @\[Parameter]
```

Note: If /uobev switch is defined in command line parameter, the SQL batch data source will not have [Pval] and [EU] available as separate columns; instead they are embedded in [Descript] column. The description of Property Template is given in the Property Template Description, below.

Property Template Description

Template Name	Allowed Placeholders in Value	Value Description

`Property[#].Name`
Or
`Attribute[#].Name`
 	
Optional
	`[TIME]`
	`[UNIQUEID]`
	`[BATCHID]`
	`[PROCEDURE]`
	`[UNITPROCEDURE]`
	`[OPERATION]`
	`[PHASE]`
	`[PHASESTATE]`
	`[PHASESTEP]`
	`[DESCRIPT]`
	`[EVENT]`
	`[PVAL]`
	`[EU]`
	`[AREA]`
	`[PROCESSCELL]`
	`[UNIT]`
	`[PHASEMODULE]`
	`[USERID]`
	`[COMMENT]`
	
	[*,value="Exact Field"],
	[*,value="Field Mask"],
	advanced parsing
		This property defines the Name structure of the PI Property. The triggering expression or Event Type must be embedded in the value structure. Allowed placeholders are not case sensitive. Due to the requirement that PI Property Names should be unique under the same PI Property
		If Template Property – Name is not defined, the PI Property names are created automatically by the interface as **`Event_(Same Node Event Count)`**.
		
		Note: Warning, if Name is defined and there is an event which results in PI Property Name already existing in PI Server, then the interface will replace the existing PI Property value with the new one.
		
		Note: Each incoming event can trigger multiple Property Templates if it is defined in each as triggering event.
		
		In Name property, the hierarchy of names is supported.
		
		Example
		`Property[1].Name = Materials\[Event]`
		
		As result, if the Property Template is triggered then the interface creates under proper Recipe PIProperty – PI Property "Materials" and as child property – the value of [Parameter] placeholder.
		
		By default all properties are placed under the particular recipe nodes in PIProperties which mimic the recipe structure created in PIBatch database.
		Starting from version 2.0.0.1, parameter data from any depth of the source recipe hierarchy can be placed at the root level of the PIBatch PIProperties object. It can be achieved by specifying the $ as the first node name in path.
		Example:
		`Property[1].Name = $\[Event]`
		
		Assume that `[Parameter]="Recipe Data".` As result, if the Property Template is triggered then the interface creates under specific root UniqueID PIProperty the property named as `"Recipe Data".`

`Property[#].Value` Or	Same as for Name. And	This property defines the value structure of the PI Property. The triggering expression or Event Type must be embedded in the

Template Name	Allowed Placeholders in Value	Value Description
`Attribute[#].Value` Required	`[TAG]`	value structure. Allowed placeholders are not case sensitive. Due to the requirement that PI Property Names should be unique under the same PI Property Node, the property names are created automatically by the interface (default). **Note:** Each incoming event can trigger multiple Property Templates if it is defined in each as triggering event. **Property Template Value:** Defined by user as free text with optional placeholders. Placeholder – is the name of the source column. For each incoming event, placeholder in the template is replaced by corresponding field from the event structure. **Note**: for SQL data source(s), if /uobev switch is used then there are no explicit columns for [Pval] and [EU], instead, these columns are combined and appended to [Descript] column. Example : Property[1].Value = [BatchID] | event: <State*> | [Descript] | val: [Pval] Or using advanced parsing definition: Property[1].Value = [BatchID] | event: [*,value=”State*”] | [Descript] | val: [Pval]

Template Name	Allowed Placeholders in Value	Value Description

`Property[#].Trigger`
`Or`
`Attribute[#].Trigger`
 	
Optional
	Same as for Name Property except
	`[TIME]`
		Defines the triggering expression or Event Type which used to create and populate PI Properties. If trigger is defined, it overrides triggering expression in Value property if defined.
		Note: There can be multiple triggers defined for a single template property.
		Note: Interface uses only placeholders embedded into expression to trigger particular template. All free text is omitted and placeholders are treated as with AND logic. Using multiple Triggering expression allows to create OR logic for triggering.
		Example 1:
		Property[1].Trigger = [Parameter, value=”State Change”]
		Property[1].Trigger = [Value, value=”test”]
		or using mask:
		Property[1].Trigger = [Parameter, value=”State Ch*”]
		Property[1].Trigger = [Value, value=”tes*”]
		Example 2:
		Using triggering expression with two placeholders:
		Tag[1].Trigger=[Event, value=”State*] [Pval,value=RUNNING”]
		This expression will trigger tag template only if both conditions are met.

`Property[#].Translate`
`Or`
`Attribute[#].Translate`
Optional
	Values:
	`true/false`
		If set to true this property allows the Property Template to use the translate map and replace words, phrases found in Value with the custom ones.

`Property[#].Type`
`Or`
`Attribute[#].Type`
 	
Optional
	`String`
	`Float`
	`Integer`
	`Auto`
		Defines the type of the PIProperty when PI server is used as the host for batch data.
		Defines the type of the AFAttribute when AF Server is used as the host for batch data.
		The Type=Auto - determines the data type of the value automatically.

`Property[#].Category`
`Or`
`Attribute[#].Category`
 	
Optional
 	
AF Only
	Same as for Name property	Defines the AF Attriubute Category property. This property can be used for grouping of the attributes.

`Attribute[#].UOM`
`Or`
`Attribute [#].EngUnits`
`Or`
	Same as for Name property	This property defines the Engineering Units (Units of Measure) for the specific AF Attribute. Allowed placeholders are not case sensitive.
		
		Note: Placeholder [EU] are not available if the

Template Name	Allowed Placeholders in Value	Value Description
`Attribute[#].EU` Optional AF Only		source is defined as SQL Server(s) and the **/uobev** switch is specified in the command line. When using the switch /uobev, the interface reads batch associated events from the batcheventview, where the [Descript] column is the union of EVT [Descript], [Pval] and [EU] columns.

EVT Example 1:

The Property Template is defined in INI file as follows:

```
Property[1].Value=[Time]:[Descript]:[Pval]:[EU]-[Event]_Testing
```

where index=1 is arbitrary (1-based) and is used to identify which template was used to create particular PIProperty event structure. The property index number must be defined in square brackets. The text string defined after the equal sign (=) is the actual PIProperty structure given in the free text format, and is not case sensitive.

Assume that incoming event from data source contains populated fields as:

```
[Time]=”12/01/2008 12:01:05”
[Descript]=abc
[Pval]=123.456
[EU]=cm
[Event]=Report
```

Then the actual PIProperty value added to the PIBatch object is:

```
12/01/2008 12:01:05:abc:123.456:cm-Report_Testing
```

SQL Example 1:

The Property Template is defined in INI file as follows:

```
   Property[2].Value=[BatchID]_[Descript]_[Event]:abc
```

where index=2 is arbitrary (1-based) and is used to identify which template was used to create particular PIProperty event structure.

There is no difference between EVT and SQL data sources unless the /uobev (Use Original Batch Event View) switch is specified. In this case [Pval] and [EU] columns are not available in SQL data source. Instead the SQL [descript] column is the union of EVT data source columns: [descript],[pval],[eu].We defined an [Event] placeholder which allows ANY incoming event to use this template and to populate PIProperties collection for each PIBatch.

Assume that incoming event from the data source contains specific fields populated as:

```
[BatchID]=Lot12345
[Descript]=abc 123.456 cm
[Event]=Report
```

Then the actual PIProperty value added to the PIBatch object is:

```
Lot12345_abc 123.456 cm_Report:abc
```

Tag Templates

The PI Interface to the Emerson DeltaV Batch has an option to store batch associated data in PI Points, commonly known as tags. Every Event Type known on the source can be recorded in the PI Server. By default, interface does not create tags or populate them with events. You can enable this functionality through the use of Tag Templates, which are defined in the INI file associated with each interface instance. The INI file should be located in the same directory as the interface startup batch file (BAT) and have the same filename (except extension) as the batch file (BAT) which is used to start the particular instance of the interface. If such setup is not possible for any reason, then INI file can be specified in optional command line parameter **`/inifile=<full path filename>`**.

Tag Templates allow defining structures for tag name, tag data type, tag value, unit alias name, phase module alias name, engunits and descriptor properties. The timestamp for each tag event is obtained directly from the data source. The required properties to be completed are tag name structure, tag value structure and tag type, all other properties are optional. If only tag name is defined then the triggering "event type" or expression should be defined as part of the tag name structure. If an explicit trigger is defined, then the tag creation and population is based on the event type or expression defined as <trigger> value overriding the tag name triggering (if defined). Multiple tag templates can be triggered by the same source event and a single template can be triggered by multiple source events.

Note: If explicit triggers are used, then the Tag Name embedded triggering is overridden.

You also have the option to specify the tag value type. There are 3 tag types allowed: float, integer and string. By default, if the value type is not specified, the batch interface creates a string type PI Point and treats all event values as strings.

Tag[index].<Property >= Free text

Where index is the 1-based positive number, also serves as Location2 value in actual PI Point attributes and is used to identify which Tag Template created specific PI Point.

Possible Tag Template <Property> definitions:

```
Tag[index].Name = Name structure (with embedded triggering Event
Type or Event Type Mask or Expression)
Tag[index].Value = Event value structure as free text
Tag[index].Trigger = Event Type or Event Type mask or Expression
Tag[index].Type   = string/integer/float/auto
Tag[index].UnitAlias = <optional sub unit module path |> unit tag
alias name structure (default: alias name as .Name and alias module
path as [Unit])
Tag[index].PhaseAlias = <optional sub phasemodule path |> phase
module  tag alias name structure (default: alias name as .Name and
alias module path as [PhaseModule])
Tag[index].Descriptor = value structure as free text (default:
blank)
Tag[index].EngUnits = value structure as free text (default: blank)
Tag[index].Translate = true/false   (default: false)
Tag[index].Annotation = free text without placeholders
Tag[index].Annotation2 = free text without placeholders
```

If the name structure contains placeholders, then the tag template will only be triggered if all placeholders are replaced with non-empty fields from each incoming event. The event value structure does not have this limitation, i.e. placeholders can be replaced with empty fields. The only exception is the advanced field value parsing. In the Tag Template Description table below, a complete list of possible properties, values and placeholders that can be used to define value/name structures is provided.

Multiple tag templates are capable of writing to the same PI tag (if the .Name attribute of the tag templates resolves to the same PI tag name). This is useful when you want different values to be written to the same PI tag dependent on the trigger for each.

Note: when **/uobev** switch is used, SQL data source does not have [Pval] and [EU] columns separately as it is in EVT data source; instead they are embedded in [Descript] column as union of EVT [Descript][Pval][EU].

Tag Template Description

Property	Allowed Placeholders	Description

`Tag[#].Name`
 	
Required
	`[UNIQUEID]`
	`[BATCHID]`
	`[PROCEDURE]`
	`[UNITPROCEDURE]`
	`[OPERATION]`
	`[PHASE]`
	`[DESCRIPT]`
	`[EVENT]`
	`[PVAL]`
	`[EU]`
	`[AREA]`
	`[PROCESSCELL]`
	`[UNIT]`
	`[PHASEMODULE]`
	`[USERID]`
	`[COMMENT]`
	
	[*,value="Exact Field"],
	[*,value="Field Mask"],
	advanced parsing,
		This property defines the name structure of the tag. Allowed placeholders are not case sensitive. The triggering "Event Type" or expression can be specified either in Tag.Name or in Tag.Trigger properties.
		The tag name structure can also contain Exact word or phrase (must be specified within angled brackets <>) which can be found in any fields of the incoming event. If resulting Tag Name contains illegal characters such as * ' ? ; { } [ ] | ` " \ then these characters are replaced by "_" character. The contained word or phrase can be also a mask.
		
		For example, Descript column contains field: B10_OP_CIP100. If it is desired to have a tag when this descriptor is encountered, then the tag name template can be specified as:
		Tag[1].Name = [unit] <B10_OP_CIP100> State Change.
		Or using masked field definition:
		Tag[1].Name = [unit] <B10_OP_CI*> State Change.
		
		Triggering event can be defined as mask as well.
		Example: Tag[1].Name = [unit] <B10_OP_CI*> <State Ch*>
		
		Note: Each incoming event can be used to create/populate multiple PI Tags, if it is defined as triggering event in multiple Tag Templates.

`Tag[#].Value`
 	
Required
	Same as for Name property and
	`[TIME]`
	`[TAG]`
		This property defines the event value structure for the specific PI Point. Allowed placeholders are not case sensitive.
		The event timestamp is taken from the incoming event [Time] field.
		
		Note: Placeholders [Pval] and [EU] are not available if source is defined as SQL Server(s) and switch **/uobev** is specified in command line. When using the switch /uobev, the interface reads batch associated events from the batcheventview, where the [Descript] column is the union of EVT [Descript], [Pval] and [EU] columns.

`Tag[#].Type`
 	
Required
	`String`
	`Float`
	`Integer`
	`Auto`
		Defines the type of the PI Point to be created and how to treat the events written to this tag.
		The Type=Auto - determines the data type of the PI Point automatically by analyzing the first event.

Property	Allowed Placeholders	Description
`Tag[#].Trigger` Optional	Same as for Name and Event Type or Event Type Mask	Defines the triggering Event Type or expression which used to create and populate PI tags. If trigger is defined, it overrides triggering Event Type or expression in Name property if defined. **Note:** There can be multiple triggers defined for a single template tag. **Note:** Interface uses only placeholders embedded into expression to trigger particular template. All free text is omitted and placeholders are treated as with AND logic. Using multiple Triggering expression allows to create OR logic for triggering. Example: Tag[1].Trigger = State Change or using mask: Tag[1].Trigger = State Ch* Using triggering expression with two placeholders: Tag[1].Trigger=[Event, value=”State*] [Pval,value=RUNNING”] This expression will trigger tag template only if both conditions are met.

Property	Allowed Placeholders	Description
`Tag[#].UnitAlias` Optional	Same as for Name property	This property defines the unit level alias name structure for specific template tag. The contained field can be defined as exact phrase or mask. If resulting Alias Name contains illegal characters such as * ‘ ? | ` “ then these characters are replaced by “_” character. Starting with interface version 1.0.1.0 optional sub unit module path can be specified in alias name. “\” symbol should be used to separate parent\child modules and “|” symbol should be used to separate module path and the actual alias name. **Default:** uses **Name** property as unit level alias name and [unit] module as alias location. **Note:** The names for PI Aliases must be unique. Starting from version 1.0.2.0, interface allows creating aliases on PI modules based on absolute module path. This can be achieved by placing the ‘$’ sign as the first module in module path. ‘$’ stands for root module. If /smp=<Start Module Path> switch is used – then ‘$’ is the leaf node of the hierarchy created from start module path. If no /smp switch in command line, then ‘$’ is the actual PI MDB root node. **Example 1**: Tag[1].UnitAlias = [phasemodule] State alias This alias is going to be created on particular [Unit] module with alias name as **[phasemodule] State alias** **Example 2:** Tag[2].UnitAlias = ABC\def | [phasemodule] State alias This alias is going to be created under **[Unit]\ABC\def** module tree with alias name template as **[phasemodule] State alias** **Example 3**: Tag[3].UnitAlias = $ \[Area] | [phasemodule] State alias In this example, assume no /smp switch is defined in command line and [Area]=”AREA_A” and [PhaseModule]=”PH101”. Then the interface is going to create hierarchy as (PI MDB) \ AREA_A And place an alias named “PH101 State alias” under AREA_A node.

Property	Allowed Placeholders	Description
`Tag[#].PhaseAlias` Optional	Same as for Name property	This property defines the phase module level alias name structure for specific template tag. The contained field can be defined as exact phrase or mask. If resulting Alias Name contains illegal characters such as *** ‘ ? | ` “** then these characters are replaced by “_” character. Starting with interface version 1.0.1.0 optional sub unit module path can be specified. “\” symbol should be used to separate parent\child modules and “|” symbol should be used to separate module path and the actual alias name. **Default:** uses **Name** property as phase module level alias name and [PhaseModule] module as alias location. **Note:** The names for PI Aliases must be unique. Starting from version 1.0.2.0, interface allows creating aliases on PI modules based on absolute module path. This can be achieved by placing the ‘$’ sign as the first module in module path. ‘$’ stands for root module. If /smp=<Start Module Path> switch is used – then ‘$’ is the leaf node of the hierarchy created from start module path. If no /smp switch in command line, then ‘$’ is the actual PI MDB root node. Example 1: Tag[1].PhaseAlias = State alias This alias is going to be created on particular **[PhaseModule]** module with name as **State alias** Example 2: Tag[2].PhaseAlias = ABC\[EVENT] | State phasemodule alias This alias is going to be created under **[Unit]\ABC\[Event]** module tree with alias name as **State phasemodule alias** Example 3: Tag[3].PhaseAlias = $ \[Area] | [phasemodule] State alias In this example, assume no /smp switch is defined in command line and [Area]=”AREA_A”. Then the interface is going to create hierarchy as (PI MDB) \ AREA_A And place an alias under AREA_A node.
`Tag[#].Descriptor` Optional	Same as for Name property	This property defines the Tag Descriptor structure for the specific PI Point. Allowed placeholders are not case sensitive.
`Tag[#].EngUnits` Optional	Same as for Name property	This property defines the Engineering Units (EngUnits) structure for the specific PI Point. Allowed placeholders are not case sensitive. **Note:** Placeholder [EU] are not available if source is defined as SQL Server(s) and switch **/uobev** is specified in command line. When using the switch /uobev, the interface reads batch associated events from the batcheventview, where the [Descript] column is the union of EVT [Descript], [Pval] and [EU] columns.

Property	Allowed Placeholders	Description
`Tag[#].Translate` Optional	Values: `true/false`	If set to true this property allows the Tag Template to use the translate map and replace words, phrases found in Name, Value, UnitAlias, PhaseAlias, Descriptor and EngUnits with the custom ones.
`Tag[#].Annotation` Optional	Same as Name property	This property enables you to annotate each value written to PI Server using the specific Tag Template. The annotation is written to PI as a string value. Example: Tag[1].Annotation = [BatchID]
`Tag[#].Annotation 2` Optional	Same as Name property	This property enables you to annotate each value written to PI Server using the specific Tag Template. The annotation is written to PI as a NameValue object. Example: Tag[1].Annotation2 = [BatchID]

EVT Example 1:

```
Tag[23].Name =[Unit] [phasemodule] [Event, value="Report"]
Tag[23].Value =[Descript]:[Pval]:[EU]-[Event]_Just for testing
Tag[23].Type = string
Tag[23].unitalias=[phasemodule] Report
Tag[23].phasealias=Report
Tag[23].Annotation=[BatchID]
```

Assume that incoming event contains the following fields populated as:

```
[BatchID]=Batch_123
[Time]=08/12/2008 13:00:05.123
[Descript]= Level
[Pval]=123.456
[EU]=cm
[Event]=Report
[Unit]=U101
[PhaseModule]=PDL123
```

Since Event Type = "Report" is embedded in Tag Name template and [Unit] and [PhaseModule] placeholders are replaced with non-empty fields, then the template is triggered. The interface is going to search for and add if necessary a PI Point named as "U101 PDL123 Report". The Interface stores the tag name in the PIPoint's Extended Descriptor field also. If the PIPoint was not known to the interface since the last startup, the interface also checks for the existence of aliases. Since the [Unit] placeholder is defined in the Tag Name, then the interface is going to find/add alias to this tag on Unit: U101 in PIModule Database as

Unit level alias name = "U101 PDL123 Report",

Because the [PhaseModule] placeholder is also defined in Tag Name, the interface finds/adds alias for this tag on PhaseModule: PDL123.

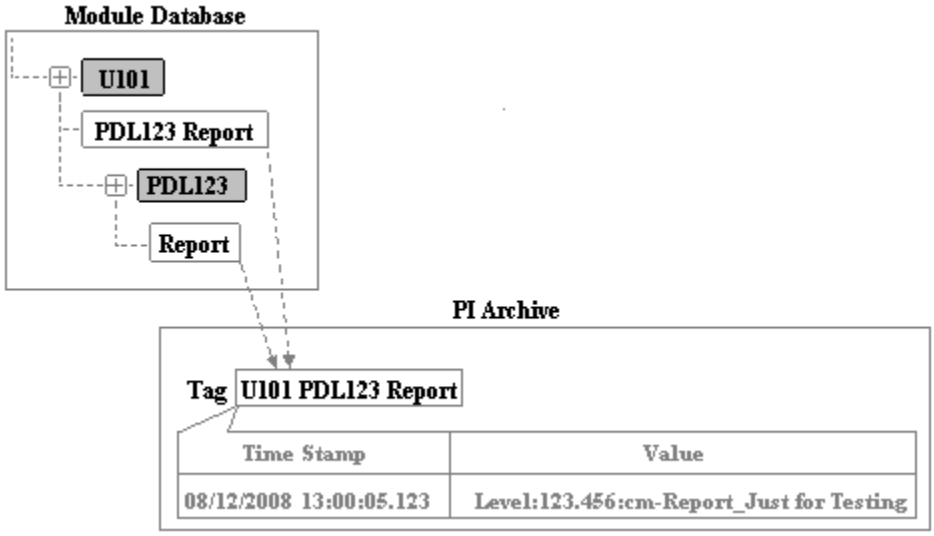
Phase level alias name = "U101 PDL123 Report,"

When PIPoint and aliases are verified the actual event value is added to the PIPoint:

Event TimeStamp	Event Value
08/12/2008 13:00:05.123	Level:123.456:cm-Report_Just for Testing

Assume that in addition to the above Name and Value properties we have defined:

Unit level Alias Name = “PDL123 Report” and the Phase Module level Alias Name = “Report” linkage to the PI Tag are illustrated in the figure below.



Because the Annotaion property is define in the tag template, the value above is annotated with the text: “Batch_123”

EVT Example 2:

Based on EVT Example 1, let’s replace the Tag Template definition for the value as

```
Tag[23].Value =[Pval]
Tag[23].Type = float
```

Assume that the incoming event contains:

[Time]=08/12/2008 13:00:05.123
[Pval]=34.12345 (as float32)

Then the resulting PIPoint named (from the EVT Example 1) as “U101 PDL123 Report” will be created or treated as a float type PIPoint, and the event is going to be sent as

Event TimeStamp	Event Value (float)
08/12/2008 13:00:05.123	34.12345

SQL Example 1:

Note: Default SQL behavior is exactly the same as EVT, example below is provided for case when SQL is selected as data source and /uobev switch is defined in command line parameters.

Based on EVT Example 1, let’s replace the Tag Template definition for the value as

```
Tag[23].Value=[BatchID]_[Descript]_[Event]:abc
Tag[23].Type = string
```

Assume that incoming event contains:

```
[BatchID]= IceCreamLot101
[Descript]=Loaded 54.3 kg
```

```
[Event]=Report
```

The resulting PIPoint named (from the EVT Example 1) as "U101 PDL123 Report" is created and treated as a string type PIPoint, and the event is sent as

Event TimeStamp	Event Value (string)
08/12/2008 13:00:05.123	IceCreamLot101_Loaded 54.3 kg_Report:abc

Note: In the Template Value definition we did not specify the value type. Therefore, by default, the interface treats PIPoint as string type *Tag*.

EVT Example 3:

Now let's consider the more complicated example involving multiple Event Types triggering the single Tag Template to find/create PIPoint and populate it with data. Multiple triggering events can be defined using keyword – **trigger**. The value for each trigger is the triggering expression, which can be defined as Event Type itself (Example below: event type – Comment) or combination of placeholders, which is treated with AND logic. Providing multiple trigger expressions as below is treated with OR logic.

```
Tag[12].Name  = [Unit] Tester
Tag[12].Value   =[pval]-[Event]
Tag[12].Type     = string
Tag[12].Trigger = [Event, value="Report"]
Tag[12].Trigger = [Event, value="State Change"]
Tag[12].Trigger = Comment
Tag[12].unitalias = Testing [Unit]
Tag[12].phasealias = Testing phase alias
```

Note: In this example we do not have the [PhaseModule] placeholder or actual phase name embedded in the Tag Template Name. Therefore, phasealias will not be created or verified.

Assume that we have 4 incoming events:

Event 1 contains the following fields populated as:

```
[Time]=08/12/2008 13:00:05.123
[Event]=Report
[Unit]=U101
[Pval]=123.456
```

Event 2 contains the following fields populated as:

```
[Time]=08/12/2008 15:20:01.998
[Event]=State Change
[Unit]=U101
[Pval]=HOLDING
```

Event 3 contains the following fields populated as:

```
[Time]=08/12/2008 18:49:12.020
[Event]=Comment
[Unit]=U101
[Pval]=Discard Product
```

and **Event 4** contains the following fields populated as:

```
[Time]=08/12/2008 21:49:14.025
```

```
[Event]=Comment
[Unit]=
[Pval]=Discard Product
```

As result, the Tag Template only gets triggered for the first 3 events. The last incoming **Event 4** does not have the [Unit] field populated; therefore the Tag Name structure is not complete and cannot be processed. All events result in tag name as "U101 Tester" and tag unit level alias name as "Testing U101".

The resulting events are stored to PI as follows:

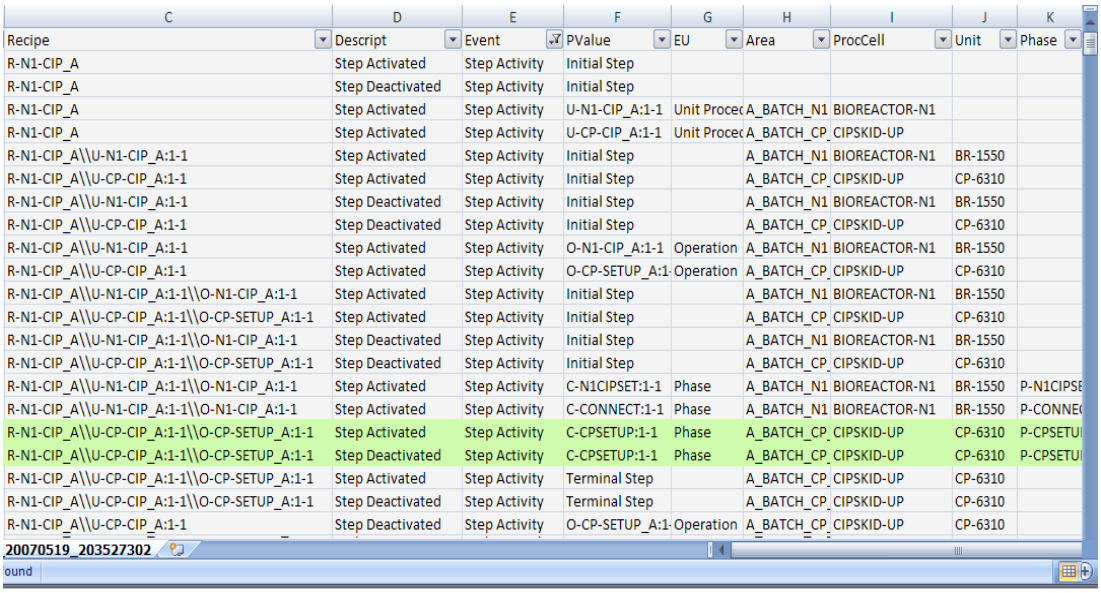
Event TimeStamp	Event Value (string)
08/12/2008 13:00:05.123	123.456-Report
08/12/2008 15:20:01.998	HOLDING-State Change
08/12/2008 18:49:12.020	Discard Product-Comment

EVT/SQL Example 4:

Now let's consider the following scenario where we want to store value "1" when Operation and Phase is started and "0" when Operation or Phase is complete. To achieve such task we can use source Event Type – "Step Activity". Then we can use single template to for both Operation and Phases to write start event, i.e. "1" and then another template to write an end event "0"

```
Tag[1].Name  = Activity on [Unit] [Pval]
Tag[1].Value   = 1
Tag[1].Type    = integer
Tag[1].Trigger = [Descript, value="Step Activated"] [Event,
value="Step Activity] [EU, value="Operation"]
Tag[1].Trigger = [Descript, value="Step Activated"] [Event,
value="Step Activity] [EU, value="Phase"]

Tag[2].Name  = Activity on [Unit] [Pval]
Tag[2].Value   = 0
Tag[2].Type    = integer
Tag[2].Trigger = [Descript, value="Step Deactivated"] [Event,
value="Step Activity] [EU, value="Operation"]
Tag[2].Trigger = [Descript, value="Step Deactivated"] [Event,
value="Step Activity] [EU, value="Phase"]
```

Assume that the current incoming events are represented by highlighted rows as in the figure below. Then the templates are going to be triggered on Step Activity for Phase module.



C	D	E	F	G	H	I	J	K
Recipe	Descript	Event	PValue	EU	Area	ProcCell	Unit	Phase
R-N1-CIP_A	Step Activated	Step Activity	Initial Step					
R-N1-CIP_A	Step Deactivated	Step Activity	Initial Step					
R-N1-CIP_A	Step Activated	Step Activity	U-N1-CIP_A:1-1	Unit Proce	A_BATCH_N1	BIOREACTOR-N1		
R-N1-CIP_A	Step Activated	Step Activity	U-CP-CIP_A:1-1	Unit Proce	A_BATCH_CP	CIPSKID-UP		
R-N1-CIP_A\\U-N1-CIP_A:1-1	Step Activated	Step Activity	Initial Step		A_BATCH_N1	BIOREACTOR-N1	BR-1550	
R-N1-CIP_A\\U-CP-CIP_A:1-1	Step Activated	Step Activity	Initial Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-N1-CIP_A:1-1	Step Deactivated	Step Activity	Initial Step		A_BATCH_N1	BIOREACTOR-N1	BR-1550	
R-N1-CIP_A\\U-CP-CIP_A:1-1	Step Deactivated	Step Activity	Initial Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-N1-CIP_A:1-1	Step Activated	Step Activity	O-N1-CIP_A:1-1	Operation	A_BATCH_N1	BIOREACTOR-N1	BR-1550	
R-N1-CIP_A\\U-CP-CIP_A:1-1	Step Activated	Step Activity	O-CP-SETUP_A:1	Operation	A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-N1-CIP_A:1-1\\O-N1-CIP_A:1-1	Step Activated	Step Activity	Initial Step		A_BATCH_N1	BIOREACTOR-N1	BR-1550	
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Activated	Step Activity	Initial Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-N1-CIP_A:1-1\\O-N1-CIP_A:1-1	Step Deactivated	Step Activity	Initial Step		A_BATCH_N1	BIOREACTOR-N1	BR-1550	
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Deactivated	Step Activity	Initial Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-N1-CIP_A:1-1\\O-N1-CIP_A:1-1	Step Activated	Step Activity	C-N1CIPSET:1-1	Phase	A_BATCH_N1	BIOREACTOR-N1	BR-1550	P-N1CIPSE
R-N1-CIP_A\\U-N1-CIP_A:1-1\\O-N1-CIP_A:1-1	Step Activated	Step Activity	C-CONNECT:1-1	Phase	A_BATCH_N1	BIOREACTOR-N1	BR-1550	P-CONNE
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Activated	Step Activity	C-CPSETUP:1-1	Phase	A_BATCH_CP	CIPSKID-UP	CP-6310	P-CPSETU
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Deactivated	Step Activity	C-CPSETUP:1-1	Phase	A_BATCH_CP	CIPSKID-UP	CP-6310	P-CPSETU
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Activated	Step Activity	Terminal Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-CP-CIP_A:1-1\\O-CP-SETUP_A:1-1	Step Deactivated	Step Activity	Terminal Step		A_BATCH_CP	CIPSKID-UP	CP-6310	
R-N1-CIP_A\\U-CP-CIP_A:1-1	Step Deactivated	Step Activity	O-CP-SETUP_A:1	Operation	A_BATCH_CP	CIPSKID-UP	CP-6310	

20070519_203527302

ound

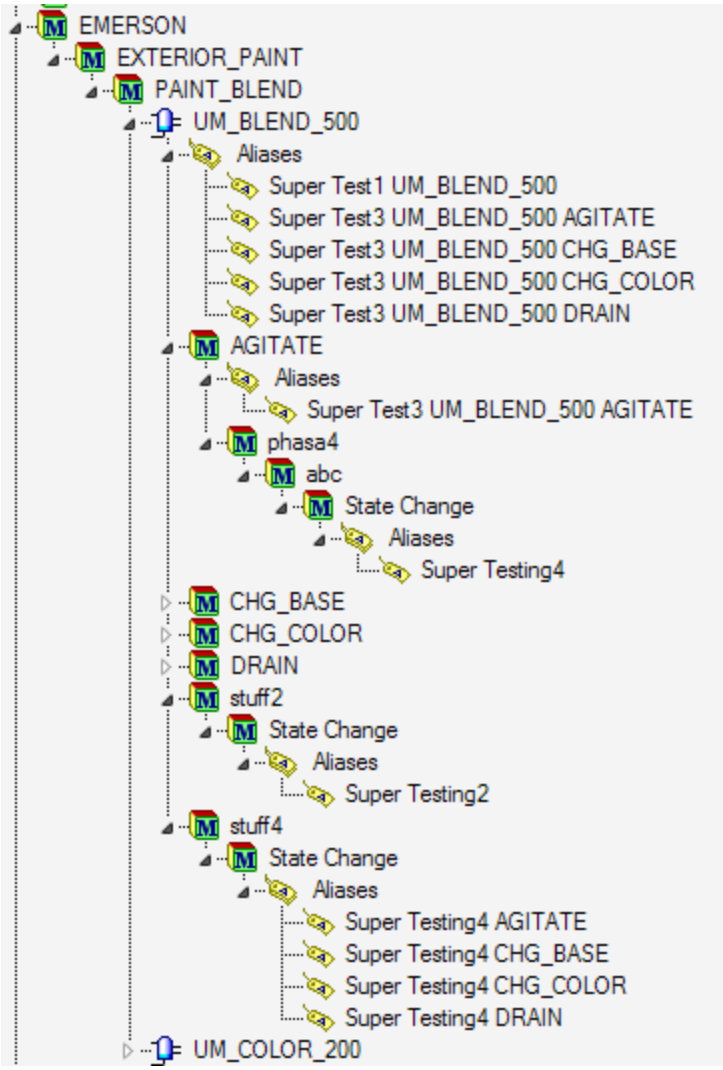
The first highlighted event triggers the template – Tag[1] with resulting PI Tag Name **"Activity on CP-6310 C-CPSETUP:1-1"** and value **"1"** is written as PI Value. The second event triggers template – Tag[2] with same resulting PI Tag Name as **"Activity on CP-6310 C-CPSETUP:1-1"** with value **"0"** is written as PI Value. Note, both templates write to the same PI Tag.

EVT/SQL Example 5:

Different variations on how PI Points can be aliased to PI Modules. Consider the following Tag Templates:

```
Tag[6].Name = Super Test1 [Unit]
Tag[6].Value = [Event] | [Pval]
Tag[6].Trigger = [Event,value=State Change]

Tag[7].Name = Super Test2 [Unit]
Tag[7].Value = [Event] | [Pval]
Tag[7].Trigger = [Event,value=State Change]
Tag[7].UnitAlias = stuff2\[Event] | Super Testing2

Tag[8].Name = Super Test3 [Unit] [PhaseModule]
Tag[8].Value = [Event] | [Pval]
Tag[8].Trigger = [Event,value=State Change]

Tag[9].Name = Super Test4 [Unit] [PhaseModule]
Tag[9].Value = [Event] | [Pval]
Tag[9].Trigger = [Event,value=State Change]
Tag[9].UnitAlias = stuff4\[Event] | Super Testing4 [PhaseModule]
Tag[9].PhaseAlias = phasa4\abc\[Event] | Super Testing4

Tag[10].Name = Super Test5 [Unit] [PhaseModule]
Tag[10].Value = [Event] | [Pval]
Tag[10].Trigger = [Event,value=State Change]
Tag[10].UnitAlias = NONE
Tag[10].PhaseAlias = NONE
```

Refer to the screenshot below for results of aliasing:


EMERSON
EXTERIOR_PAINT
PAINT_BLEND
UM_BLEND_500
Aliases
Super Test1 UM_BLEND_500
Super Test3 UM_BLEND_500 AGITATE
Super Test3 UM_BLEND_500 CHG_BASE
Super Test3 UM_BLEND_500 CHG_COLOR
Super Test3 UM_BLEND_500 DRAIN
AGITATE
Aliases
Super Test3 UM_BLEND_500 AGITATE
phasa4
abc
State Change
Aliases
Super Testing4
CHG_BASE
CHG_COLOR
DRAIN
stuff2
State Change
Aliases
Super Testing2
stuff4
State Change
Aliases
Super Testing4 AGITATE
Super Testing4 CHG_BASE
Super Testing4 CHG_COLOR
Super Testing4 DRAIN
UM_COLOR_200

Tag Templates – PI Batch Database Activity Logging

The Batch Interface is capable of providing its activity on PI Batch database by generating its own **PIEvents**. These events are based on the triggering batch event logic the interface uses against each source system to trigger PI Batches, PIUnitBatches, PISubBatches (Operations, Phases, Phase States, Phase Steps). This functionality allows customers to configure Tag Templates based on these PIEvents to write batch triggering data to PI tags (the interface is already creating PI Batch records in the PI Batch Database). PIEvent Tag Templates can be used to replace the Unit specific tag functionality (/unittags) that was available with the EVT Interface. Writing this data to PI tags allows it to be used on displays within the various PI Client Tools and reporting applications or by third party applications using data from PI tags.

PIEvent records have the following placeholders and values to be used within the .Trigger attribute of the tag template:

Placeholder	Values	Description
[EVENT]	PIEVENT	All PIEvents must trigger on [EVENT, value="PIEVENT"]
[DESCRIPT]	BATCH UNITBATCH OPERATION PHASE PHASESTATE PHASESTEP	The DESCRIPT column contains the batch level you want to trigger on. For example: [DESCRIPT, value="UNITBATCH"] Or [DESCRIPT, value="PHASE"]
[PVAL]	START END	The PVAL column contains either the start event or end event associated with the defined DESCRIPT. For example: [PVAL, value="START"] Or [PVAL, value="END"]

Multiple tag templates are capable of writing to the same PI tag (if the .Name attribute of the tag templates resolves to the same PI tag name). This is useful when you want different values to be written to the same PI tag dependent on the trigger for each. For example, a value of 1 could be written to the tag when the UnitBatch starts and a value of 0 could be written to the same tag when the UnitBatch ends.

The following placeholders are useful when writing defining the tag template (especially useful for the .Value tag template attribute):

Placeholder	Description
[BATCHID]	The Batch ID Name
[PRODUCT]	The Product Name
[PROCEDURE]	The PIBatch Procedure (Recipe) Name
[UNITPROCEDURE]	The PIUnitBatch Procedure Name
[OPERATION]	The Operation Name
[PHASE]	The Phase Name
[PHASESTATE]	The Phase State Name
[PHASESTEP]	The Phase Step Name

PIEVENT Example 1: PIBatch Active Tag

```
Tag[11].Name=BESName:PIEvent.Batch.Active
Tag[11].Value=BATCH START: [BATCHID] |Prod: [PRODUCT] |Rec:
[PROCEDURE]
Tag[11].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT, value=”BATCH”]
[PVAL,value=”START”]

//// SAME TAG
Tag[12].Name=BESName:PIEvent.Batch.Active
Tag[12].Value=BATCH END: [BATCHID] |Prod: [PRODUCT] |Rec:
[PROCEDURE]
Tag[12].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT, value=”BATCH”]
[PVAL,value=”END”]
```

PIEVENT Example 2: PIUnitBatch Active Tag

```
Tag[21].Name=BESName:[UNIT].PIEvent.UnitBatch.Active
Tag[21].Value=1
Tag[21].Type=integer
Tag[21].UnitAlias=PIEvent.UnitBatch.Active
Tag[21].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT,
value=”UNITBATCH”] [PVAL,value=”START”]

//// SAME TAG
Tag[22].Name=BESName:[UNIT].PIEvent.UnitBatch.Active
Tag[22].Value=0
Tag[22].Type=integer
Tag[22].UnitAlias=PIEvent.UnitBatch.Active
Tag[22].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT,
value=”UNITBATCH”] [PVAL,value=”END”]
```

PIEVENT Example 3: PIUnitBatch BatchID Tag

```
Tag[31].Name=BESName:[UNIT].PIEvent.UnitBatch.BatchID
Tag[31].Value=[BATCHID]
Tag[31].UnitAlias=PIEvent.UnitBatch.BatchID
Tag[31].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT,
value=”UNITBATCH”] [PVAL,value=”START”]

//// SAME TAG
Tag[32].Name=BESName:[UNIT].PIEvent.UnitBatch.BatchID
```

```
Tag[32].Value=Inactive
Tag[32].UnitAlias=PIEvent.UnitBatch.BatchID
Tag[32].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT,
value=”UNITBATCH”] [PVAL,value=”END”]
```

PIEVENT Example 4: Phase Active Tag

```
Tag[41].Name=BESName:[UNIT].PIEvent.Phase.Active
Tag[41].Value=PHASE START:
[PROCEDURE]\[UNITPROCEDURE]\[OPERATION]\[PHASE]
Tag[41].UnitAlias=PIEvent.Phase.Active
Tag[41].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT, value=”PHASE”]
[PVAL,value=”START”]

//// SAME TAG
Tag[42].Name=BESName:[UNIT].PIEvent.Phase.Active
Tag[42].Value=PHASE
END:   [PROCEDURE]\[UNITPROCEDURE]\[OPERATION]\[PHASE]
Tag[42].UnitAlias=PIEvent.Phase.Active
Tag[42].Trigger=[EVENT,value=”PIEVENT”] [DESCRIPT, value=”PHASE”]
[PVAL,value=”END”]
```

Alarm Tag Templates

The PI Interface to the Emerson DeltaV Batch has an option to store Alarm and Event data in PI Points, commonly known as tags. Every Alarm & Event stored on DeltaV SQL source can be recorded in the PI Server. By default, interface does not create alarm tags or populate them with events. You can enable this functionality through the use of Alarm Tag Templates, which are defined in the INI file associated with each interface instance. The INI file should be located in the same directory as the interface startup batch file (BAT) and have the same filename (except extension) as the batch file (BAT) which is used to start the particular instance of the interface. If such setup is not possible for any reason, then INI file can be specified in optional command line parameter **`/inifile=<full path filename>`**.

Alarm Tag Templates allow defining structures for tag name, tag data type, tag value, alias name, engunits and descriptor properties. The timestamp for each tag event is obtained directly from the data source. The required properties to be completed are tag name structure, tag value structure and tag type, all other properties are optional. If only tag name is defined then the triggering expression should be defined as part of the tag name structure. If an explicit trigger is defined, then the tag creation and population is based on the expression defined as <trigger> value overriding the tag name triggering (if defined). Multiple tag templates can be triggered by the same source event and a single template can be triggered by multiple source events.

> **Note:** If explicit triggers are used, then the Tag Name embedded triggering is overridden.

You also have the option to specify the tag value type. There are 3 tag types allowed: float, integer and string. By default, if the value type is not specified, the batch interface creates a string type PI Point and treats all event values as strings.

AlarmTag[index].<Property >= Free text

Where index is the 1-based positive number, also serves as Location2 value in actual PI Point attributes and is used to identify which Tag Template created specific PI Point.

Possible Alarm Tag Template <Property> definitions:

AlarmTag[index**].Name =** Name structure (with embedded triggering Event Type or Event Type Mask or Expression)

AlarmTag[index**].Value =** Event value structure as free text

AlarmTag[index**].Trigger** = Event Type or Event Type mask or Expression

AlarmTag[index]**.Type** = string/integer/float/auto

AlarmTag[index**].Alias =** <optional sub module path |> tag alias name structure

(default: alias name as **.Name** and alias module path as **[module]**)

AlarmTag[index**].Descriptor =** value structure as free text (default: **blank**)

AlarmTag[index**].EngUnits =** value structure as free text (default: **blank**)

AlarmTag[index].Translate = true/false (default: false)

If the name structure contains placeholders, then the tag template will only be triggered if all placeholders are replaced with non-empty fields from each incoming event. The event value structure does not have this limitation, i.e. placeholders can be replaced with empty fields. The only exception is the advanced field value parsing. In the Tag Template Description table below, a complete list of possible properties, values and placeholders that can be used to define value/name structures is provided.

AlarmTag Template Description

Property	Allowed Placeholders	Description

`AlarmTag[#].Name`
 	
Required
	`[EVENT]`
	`[CATEGORY]`
	`[NODE]`
	`[AREA]`
	`[PROCESSCELL]`
	`[UNIT]`
	`[MODULE]`
	`[MODULEDESC]`
	`[ATTRIBUTE]`
	`[STATE]`
	`[LEVEL]`
	`[DESC1]`
	`[DESC2]`
	<Exact Field> or [*,value="Exact Field"],
	<Field Mask> or [*,value="Field Mask"],
	advanced parsing,
		This property defines the name structure of the tag. Allowed placeholders are not case sensitive. The triggering expression can be specified either in AlarmTag.Name or in AlarmTag.Trigger properties.
		The tag name structure can also contain Exact word or phrase (must be specified within angled brackets <>) which can be found in any fields of the incoming event. If resulting Tag Name contains illegal characters such as * ' ? ; { } [ ] | ` " \ then these characters are replaced by "_" character. The contained word or phrase can be also a mask.
		
		For example Category column contains field: PROCESS. If it is desired to have a tag when this category is encountered, then the tag name template can be specified as:
		Tag[1].Name = [module] [Category,value="PROCESS"]
		
		Or using masked field definition:
		
		Tag[1].Name = [module] [Category,value="PROC*"]
		
		Note: Each incoming event can be used to create/populate multiple PI Tags, if it is defined as triggering event in multiple Tag Templates.

Property	Allowed Placeholders	Description

`AlarmTag[#].Value`
 	
Required
	Same as for Name property and
	`[TIME]`
	`[TAG]`
		This property defines the event value structure for the specific PI Point. Allowed placeholders are not case sensitive.
		The event timestamp is taken from the incoming event [Time] field.

`AlarmTag[#].Type` Required	String Float Integer Auto	Defines the type of the PI Point to be created and how to treat the events written to this tag. The Type=Auto - determines the data type of the PI Point automatically by analyzing the first event.
`AlarmTag[#].Trigger` Optional	Same as for Name property	Defines the triggering expression which used to create and populate PI tags. If trigger is defined, it overrides triggering expression in Name property if defined. **Note:** There can be multiple triggers defined for a single template tag. **Note:** Interface uses only placeholders embedded into expression to trigger particular template. All free text is omitted and placeholders are treated as with AND logic. Using multiple Triggering expression allows to create OR logic for triggering. Example: Tag[1].Trigger = [Attribute, value="HI*"] Using triggering expression with two placeholders: Tag[1].Trigger=[Event, value="Alarm"] [Attribute, value="HI*"] This expression will trigger tag template only if both conditions are met.
`AlarmTag[#].Alias` Optional	Same as for Name property	This property defines the unit level alias name structure for specific template tag. The contained field can be defined as exact phrase or mask. If resulting Alias Name contains illegal characters such as * ' ? | ` " then these characters are replaced by "_" character. Optional sub [module] path can be specified within alias name. "\" symbol should be used to separate parent\child modules and "|" symbol should be used to separate module path and the actual alias name. **Default:** uses **Name** property as [**module**] level alias name. **Note:** The names for PI Aliases must be unique. Starting from version 1.0.2.0, interface allows creating aliases on PI modules based on absolute module path. This can be achieved by placing the '$' sign as the first module in module path. '$' stands for root module. If /smp=<Start Module Path> switch is used – then '$' is the leaf node of the hierarchy created from start module path. If no /smp switch in command line, then '$' is the actual

Property	Allowed Placeholders	Description
		PI MDB root node. Example 1: Tag[1].Alias = [module] State alias This alias is going to be created on particular [module] module with alias name based on template: **[module] State alias** Example 2: Tag[2].Alias = ABC\def | test[module] This alias is going to be created under [**module]\ABC\def** module tree with alias name base d on template: **test[module]** Example 3: Tag[3].Alias = $ \[Area] | [module] ALARM alias In this example, assume no /smp switch is defined in command line and [Area]="AREA_A". Then the interface is going to create hierarchy as (PI MDB) \ AREA_A And place an alias under AREA_A node.
`AlarmTag[#].Descriptor` Optional	Same as for Name property	This property defines the Tag Descriptor structure for the specific PI Point. Allowed placeholders are not case sensitive.
`AlarmTag[#].EngUnits` Optional	Same as for Name property	This property defines the Engineering Units (EngUnits) structure for the specific PI Point. Allowed placeholders are not case sensitive.
`AlarmTag[#].Translate` Optional	Values: `true/false`	If set to true this property allows the AlarmTag Template to use the translate map and replace words, phrases found in Name, Value, Alias, Descriptor and EngUnits with the custom ones.

Example 1:

Store all Alarm and Events coming from DeltaV SQL Alarm and Events historian with automatic module generation and auto alias creation

```
AlarmTag[1].Name = Alarm Test [module]
AlarmTag[1].Value = [Event] | [Category] | [Area] | [ProcessCell] |
[Unit] | [Module] | [ModuleDesc] | [Attribute] | [State] | [Level] |
[Desc1] | [desc2]
AlarmTag[1].trigger = [Event]
```

Example 2:

Store in a single global Alarm tag ONLY Alarm and Events with Event Level >= 10 or Event Level =5

```
AlarmTag[2].Name = Alarm Test Global
AlarmTag[2].Value = [Event] | [Category] | [Area] | [ProcessCell] |
[Unit] | [Module] | [ModuleDesc] | [Attribute] | [State] | [Level] |
[Desc1] | [desc2]
AlarmTag[2].trigger = [Event] [level, value="1#*"]
AlarmTag[2].trigger = [Event] [level, value="5*"]
```

Example 3:

Create integer tag for each Module and Category combination. Using advanced parsing store only numerical values embedded into the Desc2 column (Example: Desc2 contains: "VALUE = 12" and we want to store only numerical value "12") for Event type: "CHANGE" and Attribute column containing "ALM_COUNTER" substring (using mask).

AlarmTag[3].Name = Test [module] [Category]
AlarmTag[3].Value = [desc2,lbe="VALUE = "]
AlarmTag[3].type = integer
AlarmTag[3].Trigger = [Event, value="CHANGE"] [attribute, value="ALM_COUNTER*"]

PI Tag as Placeholder

The interface allows having existing host PI Tags as input data sources. Based on batch event triggering mechanism the interface can query data from PI Tags (defined on host PI server) and write results into new data structures defined by Tag and Property Templates below. The allowed syntax:

[Tag, Name="PI Tag Name", <*list of parameters delimited by comma***>**

Parameter	Description
Name="string" Required	Defines the exact name of the PI Tag which should be used for data retrieval.
Range="substring" Optional	Defines the time frame for which the data should be queried. It can be number of events, time frame or "PIOBJECT". "PIOBJECT" instructs the interface to use the time frame of the related PI batch/unitbatch/subbatch object Examples: Range="10" - the last ten events from triggered batch event timestamp are going to be retrieved. Range="10d" – the events for last 10 days from the triggered batch event timestamp are going to be retrieved. Range="PIOBJECT" – the events are going to be retrieved for the time frame of the related batch object start and end times.
Func="substring" **Optional**	Should be used in conjunction with Range parameter. It defines the aggregation function to be used on retrieved data. Possible values for this parameter: "MIN" - calculate minimum value over the range. "MAX" - calculate maximum value over the range. "TOTAL" – calculate summary of values over the range. "MID" – calculate average of values over the range.

The advanced parsing parameters can be used in [Tag] placeholder as well.

Property Template Example:

```
Property[1].Name = TestTagCalc
Property[1].Value = total:[Tag, name="sinusoid", range="10d",
func="TOTAL"] and min:[Tag, name="test_data_1", range="10d",
func="MIN"]
Property[1].Trigger = [Event,value="PIEVENT"]
[Descript,value="BATCH"] [Pval,value="START"]
```

In this example the Property Template is triggered on internal event thrown when the PI Batch is Created (Started). This template creates PI Property under the created batch with

name “TestTagCalc” and value as string data from two tags: “sinusoid” and “test_data_1”. Assuming that for time range – [(batch start) – 10d - (batch start)] summation of event values for PI Tag “sinusoid” is 1000, and for the same range the minimum for tag “test_data_1” is -25.123, then the resulting name and value combination written to PI Batch Properties is:

```
TestTagCalc = total:1000 and min:-25
```

Tag Template Example 1:

```
Tag[1].Name = Global Tester 1
Tag[1].Value = [Tag,name=”test4_data”, range=”10d”, func=”total”]
Tag[1].Trigger = [Event,value=”PIEVENT”] [Descript, value=”BATCH”]
[Pval, value=”START”]
```

In this example the Tag Template is triggered on internal event thrown when the PI Batch is Created (Started). The result written to PI Tag named “Global Tester 1”. Assuming that for time range: [((batch start) – 10d) - (batch start)] summation of event values for PI Tag “test4_data” is 1234, then the resulting value written to PI Tag:

```
Timestamp: (batch start)
Value:     1234
```

Tag Template Example 2:

```
Tag[2].Name = Global Tester 2
Tag[2].Value = [Tag,name=”test2_data”, range=”PIOBJECT”,
func=”total”]
Tag[2].Trigger = [Event,value=”PIEVENT”] [Descript, value=”BATCH”]
[Pval, value=”START”]
```

Similar to Tag Template Example 1, only the result is calculated based on time range: [(batch start) – (batch end)].

Tag Template Example 3:

```
Tag[3].Name = Global Tester 3
Tag[3].Value = [Tag,name=”test2_data]
Tag[3].Trigger = [Event,value=”State Change”] [Descript,
value=”running”]
```

In this example the Tag Template is triggered on Siemens batch event “State Change” with descriptor field as “RUNNING”. The resulting tag name is “Global Tester 3” and the value is taken from PI Tag “test2_data” at time stamp of the Siemens batch event.

Recipe Templates

Starting with version 2.0.0.1, the recipe template functionality is expanded, now all properties of the PI objects can be redefined during synchronization. Recipe templates allow redefining the recipe name convention used for PIBatch, PIUnitBatch and PISubbatch object definitions.

Syntax:

```
Recipe[index].Name= Free text
Recipe[index].BatchID = Free text
Recipe[index].ModulePath = Free text defining path
Recipe[index].Product = Free text
Recipe[index].ProductTrigger = Triggering expression
```

```
Recipe[index].Translate= true/false or 1/0
Recipe[index].Merge = true/false or 1/0

Recipe[index].Category = Free text
Recipe[index].Category[Index2].Name = Free text
Recipe[index].Category[index2].Trigger = Free text

Recipe[index].Template = Free text
Recipe[index].Template[Index2].Name = Free text
Recipe[index].Template[index2].Trigger = Free text

Recipe[index].Attribute[index2] – Enables you to define an
attribute template for specific Recipe level. See Attribute / Property
Templates for details about configuration options.
```

where 'Index' is defined as the depth (level) in recipe hierarchy. Index is a 1-based number. The possible placeholders which can be used in Name template are listed in the table below.

Template Name	Placeholders Allowed in Value	Value Description

`Recipe[#].Name`
 	
Required
	`[UNIQUEID]`
	`[BATCHID]`
	`[PROCEDURE]`
	`[UNITPROCEDURE]`
	`[OPERATION]`
	`[PHASE]`
	`[PHASESTATE]`
	`[PHASESTEP]`
	`[DESCRIPT]`
	`[EVENT]`
	`[PVAL]`
	`[EU]`
	`[AREA]`
	`[PROCESSCELL]`
	`[UNIT]`
	`[PHASEMODULE]`
	`[USERID]`
	[*,value="Exact Field"],
	[*,value="Field Mask"],
	advanced parsing
		This property defines the naming convention used by the interface to create PIBatch, PIUnitBatch and PISubbatch objects. The '#' defines the level (depth) in recipe hierarchy.
		
		Currently supported recipe levels (Index in Recipe template):
		1 – Procedure (PIBatch Recipe field)
		2 – Unit Procedure (PIUnitBatch Procedure)
		3 – Operation (PISubBatch Name field)
		4 – Phase (PISubBatch Name field)
		5 – Phase State (PISubBatch Name field)
		6 – Phase Step (PISubBatch Name field)
		
		Defaults:
		Recipe[1].Name=[Procedure]
		Recipe[2].Name = [UnitProcedure]
		Recipe[3].Name=[Operation]
		Recipe[4].Name=[Phase]
		Recipe[5].Name=[PhaseState]
		Recipe[6].Name=[PhaseStep]
		
		Example
		`Recipe[1].Name = abc_[Procedure]`
		Assume that the incoming event contains field [Procedure] with the value "Test". As result, the PIBatch Recipe field is going to be "abc_Test" instead of default "Test".

`Recipe[#].BatchID` Optional	Same as Name	This property enables you to specify the BatchID of the particular Recipe object. Currently supports PIBatch BatchID field and PIUnitBatch BatchID field.
`Recipe[#].Descriptor`	Same as Name	This property enables you to specify the AF event frame Descriptor property for the particular source

Template Name	Placeholders Allowed in Value	Value Description
Optional AF Only		Recipe object.
Recipe[#].ModulePath Optional	Same as Name	This property enables you to specify the Module path in the PI MDB, or the element path in the PI AF database, of the particular Recipe. Currently supports PIUnitBatch level (2) only. Note, the end module in the path is ALWAYS treated as Unit and will be marked in PIMDB as PIUnit.
Recipe[#].Product Optional	Same as Name	This property enables you to specify the Product of the particular Recipe object. Currently supports PIBatch Product field and PIUnitBatch Product field. Note: If ProductTrigger is not defined, this template is populated based on the data which exists in the event which creates the particular Recipe Object. Example: Recipe[1].Product =[Product] In this case the interface looks for the placeholder called [Product] to be available in the source event which creates PIBatch object
Recipe[#].ProductTrigger Optional	Same as Name	This property allows populating the Product field of the particular Recipe object after the object is created. This is useful when the Product is defined as a separate event. Example: Recipe[1].Product = [Value] Recipe[1].ProductTrigger = [Parameter, Value="Recipe Header"] [Descript, value="Product Name"]
Recipe[#].Translate Optional	True (1) False (0)	This property forces the interface to check the resulting Names against translation table. Default: False
Recipe[#].Merge Optional	True (1) False (0)	This property allows interface to merge same named objects under the same parent. Default: False
Recipe[#].Category Optional AF Only	Same as Name	For each recipe level, This property enables you to define the AF EventFrame Category. If the event that creates an event frame contains insufficient information, no category is assigned. To assign Category to an event frame after its

Template Name	Placeholders Allowed in Value	Value Description
		creation, use the dynamic Category[x] property
Recipe[#].Category[x].Name Optional AF Only	Same as Name	For each recipe level, this dynamic property enables you to define the AF event frame Category based on any event that is related to particular recipe item. This property can create as many categories as desired. Index: 'x' can be any positive integer that binds Name and Trigger(s) subproperties for specific Category[x] property. If AF Category does not exist on the AF Server, the interface creates it automatically. This property must be used with Recipe[#].Category[x].Trigger Example: Recipe[1].Category[10].Name = SCR Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]
Recipe[#].Category[x].Trigger Optional AF Only	Same as Name	This property defines the triggering expression for specific Reci[pe[#].Category[x]. There can be multiple triggers per single Recipe[#].Category[x].Name. This property must be used with Recipe[#].Category[x].Name Example: Recipe[1].Category[10].Name = SCR Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"] Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_01"] Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_02"]
Recipe[#].Template Optional AF Only	Same as Name	For each recipe level, This property enables you to define the AF event frame Template. If sufficient information is not available in the event that creates an event frame, then no AF Template is assigned. To assign Template to an event frame after its creation, use the dynamic Template[x] property
Recipe[#].Template[x].Name Optional AF Only	Same as Name	For each recipe level, this dynamic property enables you to define the AF event frame Template. based on any event that is related to particular recipe item. This property can assign only one AF template to a particular AF event frame. The interface uses the first matching Recipe[#].Template[x] property to be assigned to an event frame. Index: 'x' can be any positive integer that binds Name and Trigger subproperties for Template[x] property. This property must be used with Recipe[#].Template[x].Trigger

Template Name	Placeholders Allowed in Value	Value Description
		Example: Recipe[1].Template[10].Name = BATCH_A Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]
Recipe[#].Template[x].Trigger Optional AF Only	Same as Name	This property defines the triggering expression for the AF event frame Template. There can be multiple triggers for a single Recipe[#].Template[x]. This property must be used with Recipe[#].Template[x].Name Example: Recipe[1].Template[10].Name = BATCH_A Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"] Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_01"] Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_02"]

Default (precompiled) Recipe Templates:

```
// batch
Recipe[1].Name = [Procedure]
Recipe[1].BatchID = [BatchID]
Recipe[1].Product = [pval]
Recipe[1].ProductTrigger = [Event,value=”Recipe Header”]
[Descript,value=”Product Code”]

(AF Only)
Recipe[1].Category = OSIBatch
Recipe[1].Template = Procedure

Recipe[1].Attribute[1].Name  = [Descript]
Recipe[1].Attribute[1].Value = [Pval]
Recipe[1].Attribute[1].UOM   = [EU]
Recipe[1].Attribute[1].Category = Recipe Header
Recipe[1].Attribute[1].Trigger  = [Event, value=”Recipe Header”]

Recipe[1].Attribute[2].Name  = [Descript]
Recipe[1].Attribute[2].Value = [Pval]
Recipe[1].Attribute[2].UOM   = [EU]
Recipe[1].Attribute[2].Category = Formula Header
Recipe[1].Attribute[2].Trigger  = [Event, value=”Formula Header”]

// unitbatch
Recipe[2].Name = [UnitProcedure]
Recipe[2].BatchID = [BatchID]
Recipe[2].Product = [pval]
Recipe[2].ProductTrigger = [Event,value=”Recipe Header”]
[Descript,value=”Product Code”]
```

```
Recipe[2].ModulePath = [Area]\[ProcessCell]\[Unit]
(AF Only)
Recipe[2].Category = OSIBatch
Recipe[2].Template = UnitProcedure

// operation
Recipe[3].Name = [Operation]
(AF Only)
Recipe[3].Category = OSIBatch
Recipe[3].Template = Operation

// phase
Recipe[4].Name = [Phase]
Recipe[4].ModulePath = [Area]\[ProcessCell]\[Unit]\[PhaseModule]
(AF Only)
Recipe[4].Category = OSIBatch
Recipe[4].Template = Phase

// phase state
Recipe[5].Name = [PhaseState]
(AF Only)
Recipe[5].Category = OSIBatch
Recipe[5].Template = PhaseState

// phase step
Recipe[6].Name = [PhaseStep]
(AF Only)
Recipe[6].Category = OSIBatch
Recipe[6].Template = PhaseStep
```

Recipe Template Example 1:

```
Recipe[1].Name = [Procedure]_[UniqueID]
Recipe[3].Name = [UnitProcedure]_[Operation]
```

In this example Recipe Templates redefined PIBatch Recipe to be the concatenation of data source Procedure field + UniqueID fields and PISubbatch – Operation Name to be the concatenation of data source UnitProcedure field + Operation fields.

Recipe Template Example 2:

```
Recipe[2].Category = CAT_UNITBATCH
```

In this example, the Recipe Template defines the static PI AF Category "CAT_UNITBATCH" for the UnitProcedure level. If missing, this category is created and assigned to all UnitProcedure-level event frames regardless of the recipe type.

Recipe Template Example 3:

```
Recipe[1].Category[1].Name    = PROC_A
Recipe[1].Category[1].Trigger = [Procedure, value="DVProc:1-*"]
Recipe[1].Category[2].Name    = [Pval]
Recipe[1].Category[2].Trigger = [Descript, value="Product Code"]
```

In this example, the Recipe Template defines the dynamic PI AF Category for root level AF event frame based on source Procedure or Product Code values. If the AF Category is not found on the AF Server, the interface creates it.

Recipe Template Example 4:

```
Recipe[2].Template = OSI_UnitProcedure
```

In this example, the Recipe Template defines the static PI AF Template "OSI_UnitProcedure" for the UnitProcedure level. If missing, the PI AF Template is created and assigned to all UnitProcedure-level event frames regardless of the recipe type.

Recipe Template Example 5:

```
Recipe[2].Template[1].Name    = UP_A
Recipe[2].Template[1].Trigger = [UnitProcedure, value="UProc:1-*"]
Recipe[2].Template[2].Name    = UP_B
Recipe[2].Template[2].Trigger = [UnitProcedure, value="UProc:2-*"]
```

In this example, the Recipe Template defines the dynamic PI AF Template for the UnitProcedure-level AF event frame based on the name of the UnitProcedure. If the AF Template is not found on the AF Server, the interface creates it.

Merging Multiple Source batches into a Single PIBatch

The PI Interface to the Emerson DeltaV Batch has the ability to merge multiple source batches into one single PIBatch. This feature is enabled by using the **/merge** parameter in command line parameters. Source batches with the same BatchID are merged into one PIBatch. When a new batch is found on the source, the interface locates the identical batch in the local batch cache and adds the new batch to the existing one. The**/cachetime** parameter (default: 1day) specifies the duration in days (can also be a fraction of the day) for which the interface keeps the closed batches in the local memory.

> **Note:** The interface only merges batches which are within cached time frame, i.e. cached in the local memory.

If the batch with the identical BatchID was not found, the interface creates a new one. The **/bidm** parameter is optional and allows the interface to use a substring of the source batch BatchID as the BatchID for the PIBatch. See Using /BIDM Parameter below for details on how this switch works. Regardless of the use of the **/bidm** parameter, unitbatches under merged batch always contain the original BatchID, Recipe and Product. Each merged batch stores its original information such as full BatchID, Product, Recipe, Formula, Start and End times in the PI Properties of the merged batch under the PIProperty node named as the source batch UniqueID. Event logging in merged a PIBatch is identical to the mode when merging is not used.

Using /BIDM Parameter

The **/bidm** (**B**atch**ID M**ask) parameter is used to obtain a new BatchID, which is a substring of the value in the BatchID column in the data source. As a value the **/bidm** takes a list of BatchID masks, where the order of importance depends on the position of the mask in the list. The mask can consist of an array of valid symbols and/or wildcards. The following table represents available wildcards which can be specified within the BatchID mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Example for **/bidm** parameter to extract a substring from the BatchID column in the data source:

Let's say that the BatchID column contains: **lot30112 / 90dev123 / 12345stp / ld567.**

If **/bidm=#####** is defined then there are 5 contiguous digits and no characters in the substring. Since there are two matches, the first substring is used and the result is **30112**.

If **/bidm=######** is defined then there are 6 contiguous digits and no characters in the substring and there is no match for this and the complete string **lot30112 / 90dev123 / 12345stp/ld567** is used as the BatchID.

If **/bidm=###** is defined then there are 3 contiguous digits and no characters in the substring. Since there are two matches, the first substring is used and the result is **123**.

If **/bidm=@@@#####** is defined then there are 5 contiguous digits with 3 contiguous characters and the characters are placed before the sequence of digits. Hence the resulting BatchID is **lot30112**.

If **/bidm=##@@@###** is defined then there are 5 digits with 3 contiguous characters and the characters are placed before the third digit. Hence the resulting BatchID is **90dev123**.

If **/bidm=#####@@@** is defined then there are 5 contiguous digits with 3 contiguous characters and the characters are followed the digits. Hence the resulting BatchID is **12345stp**.

If **/bidm=?????** Is defined then any sequence of 5 symbols so the PIBatch BatchID is **lot30**.

Linking BES to MES batches

Starting with version 2.1.0.x, the interface enables you to link (bind) Batch Execution Systems batches and Manufacturing Execution System batches. This feature enables you to use PI event frames to build an integrated structure that records the master-slave relationship between the Manufacturing Execution System and the subordinate Batch Execution System.

This functionality is available only when the interface is configured to write batch data to event frames. For example, in an environment where an MES System (Emerson Syncade or Werum PAS|X) is launching BES Batches (Emerson DeltaV or Rockwell FactoryTalk Batch, the EMDVB interface always acts as an MES child when configured for lining. For each BES

(DeltaV) batch, it attempts to find the corresponding parent MES batch event frame that triggered the BES execution. The MES interface always acts as a parent and, if configured for linking, it attempts to find a child BES batch event frame when the event triggering the BES execution is acquired.

Because the MES and BES interfaces work asynchronously, they require a common element to resolve linkage. To configure this common element, specify the following command line parameter in the startup batch file for both the MES and BES interfaces:

```
/link=<element path>
```

Although an MES batch can be accessed only as the root event frame, the BES batch can be accessed asboth the root event frame and as the child of the MES event frame that represents the MES processing step that triggered the BES execution.

Lost Connections to PI Server and PI Archive Backup Issues

The interface is designed to detect and recover from connection loss from either the data sources or the PI Server, or both. If the connection is lost during processing, the Interface suspends all actions until the PI and data sources are available for communications. If the data source connection is down, the interface retries to connect on every scan until it succeeds. In configurations where SQL and OPCAE data sources are used simultaneously, connection loss to either OPCAE or SQL server(s) would not prevent the interface from receiving data from a good source. If connection only to the OPCAE server was lost and reestablished, then OPCAE data will be considered as valid only after the next SQL scan is performed to ensure that there are no data gaps while the OPCAE server was disconnected. If the PI server connection is down, the interface attempts to reconnect every **/retry** (default: 60) seconds until the **/retryTO** (default: 0- which is infinity) timeout is reached. Connection to the data sources or the PI Server could be lost for various reasons including broken physical network, data source shutdown, PI Server or required Server's subsystem shutdown, PI Server backup, freeze, etc. The Interface logs the errors to the local **pipc.log** file.

During interface shutdown and restart no data is lost. All data is buffered by the data sources. If the Interface is interrupted and it did not finish processing data from the source to the PI Server, it saves the last good processed event timestamp on shutdown. On each startup, the Interface continues processing from this timestamp in recovery mode with later switching to real-time mode.

Data Preprocessing

The PI Interface to the Emerson DeltaV Batch is designed to handle situations when the source data needs to be written to PI archives which are earlier than the primary PI archive. Due to the nature of the PI Server, the newly added tags, units and modules are indexed (referenced) only in the primary PI archive. Any older archive does not have knowledge of these modules, units and tags. In **`Preprocess`** mode the interface creates only modules, units, tags and tag aliases without processing batch data and adding events into the tags. On completion, the interface stops and the user has to reprocess older archives with the offline archive utility.

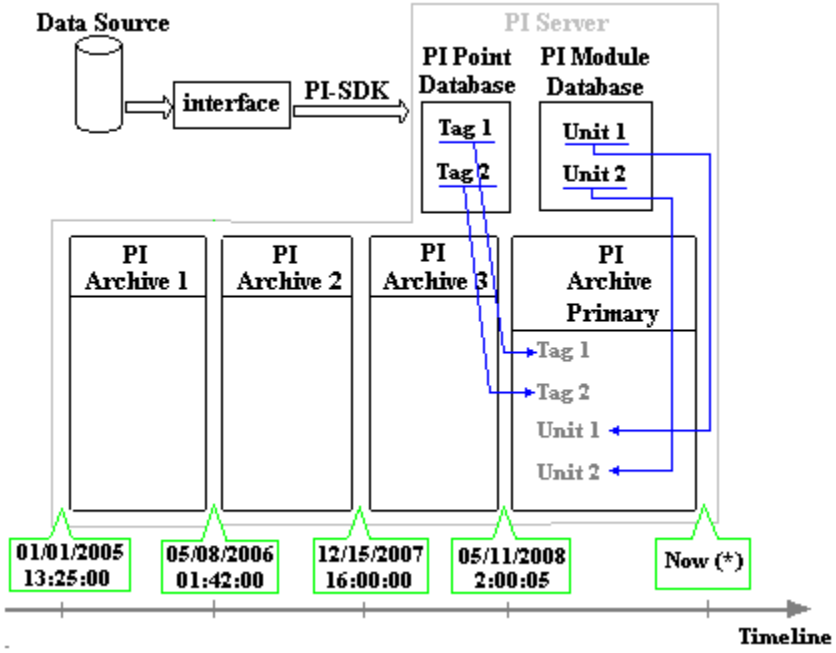
Note: The PI server does not allow any data to be written to older archives unless each older archive knows about the units and tags of interest. Refer to the *PI Server System Management Guide* for details on the archive reprocessing procedure. Note that this procedure has changed in PI Server 2012.

Reprocessing creates indexes for newly added units, modules, tags in each reprocessed archive.

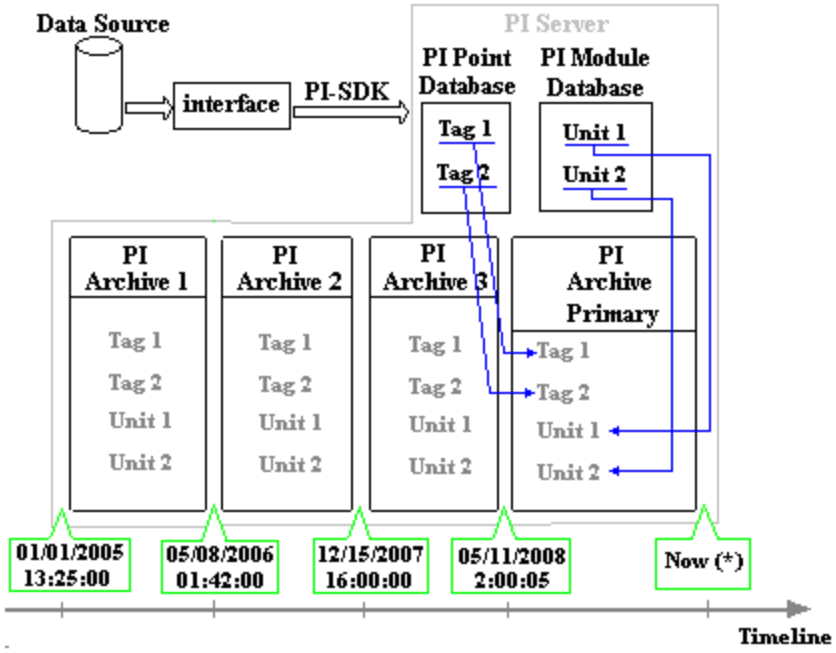
This mode should be always used before writing new batch data to older PI archives. This mode is enabled by simply adding the **/mode=NoData** parameter in conjunction with the recovery start time (**/rst**) and optional recovery end time (**/ret**) parameters to command line parameters.

Example:

Consider the time range for recovery – [**01/15/2005 12:00:00 – 06/20/2008 13:00:00**].



In the figure above the interface was run in `Preprocess` mode, where only tags and units were created in **PIPoint** and **PIModule** databases with references in the **Primary** archive only. After reprocessing **PI Archive 1**, **2** and **3** with PI Archive offline utility (**piarchss**), the **PI archives 1**, **2** and **3** now contain references to the newly created tags and units as shown in the figure below.



At this point the interface can be run in **`Recovery`** mode (using only **/rst** and optional **/ret** parameters) to backfill data into PI Points and PI Batch database.

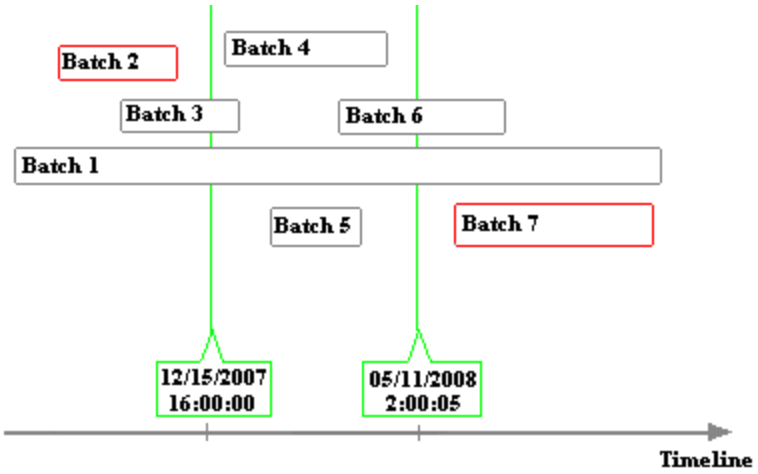
Data Recovery

The Batch interface can perform recovery of historical data. The **`Recovery`** mode of the interface can be used to perform recovery based on the clean PI archives or recover missing data for already existing PIModule, PIBatch and PIPoint objects. These objects include: PI modules, PI units, unit level aliases, phase level aliases, PIBatches, PIUnitBatches, PISubbatches (Operations, Phases, Phase States and Phase Steps), PIProperties, PIPoints, PIPoint events. When a PI object exists but contains incorrect data comparing to the source, the interface attempts to correct the PI object to match the data from the source. In the worst case, the interface prints the error message to the **PIPC.log** file. In such case the PI server needs to be cleaned first by using **`/mode=delete`**, then the recovery steps have to be performed again.

In **`Recovery`** mode, all open batches are processed only when there are no completed batches left to be processed, i.e. the interface reached the current time. If the Interface starts in **`Recovery`** mode without defining the Recovery End Time (parameter **`/ret=<date and time>)`**, it prints the results of the recovery process and changes to **`RealTime`** mode as soon as it reaches current time. Recovery mode is enabled when the Recovery Start Time parameter **`(/rst=<date and time>)`** is specified in command line parameters. The Recovery End Time parameter (**/ret**) is optional and has no effect without the **`/rst`** parameter.

> **Note:** If you specify the Recovery End Time parameter, the interface stops on completion.

For example consider recovering data from 12/15/2007 16:00:00 through 05/11/2008 2:00:05. Assume that 7 batches exist on the data source as shown in figure below:



To perform recovery, you must specify the following command line parameters:

/rst="12/15/2007 16:00:00" and **/ret="05/11/2008 2:00:05".**

These parameters are sufficient to perform historical data recovery process. In this case, the interface recovers contained batches (Batch 4 and 5) as well as border batches (Batch 1, Batch 3 and Batch 6). Batches outside the time frame (Batch 2 and 7) will NOT be recovered.

If you require recovery from 12/15/2007 16:00:00 until now (*), specify the following command line parameters:

/rst="12/15/2007 16:00:00"

> **Note:** In this case, the interface recovers (Batch 7) as well and continues processing in **RealTime**.

Data Analysis

The Batch interface can perform data analysis by comparing source data with the PI server data. In this mode the interface does not write or modify any data on the PI Server. Upon completion the interface reports results to **PIPC.log** file and stops. To enable this mode, the parameter **(/mode=stat)** must be specified in command line parameters. Data analysis can be performed only when recovery start **(/rst)** and optional recovery end time **(/ret)** are specified, otherwise the data analysis will be performed from the last processed event timestamp until system current time.

For example consider the example and figure provided in the **Data Recovery** section. If it is required to analyze data between data source(s) and PI server, then the sufficient command line parameters should be:

/rst="12/15/2007 16:00:00" /ret="05/11/2008 2:00:05" /mode=stat

If you require analysis from 12/15/2007 16:00:00 until now (*), then command line parameters should be:

/rst="12/15/2007 16:00:00" /mode=stat

PI Data Deletion

The Batch interface can perform selective data deletion stored in PI server based on the source data.

> Note: The interface cleans PI archives based on specified source data only, leaving data from all other data sources intact. This mode should be used only if the interface is unable to synchronize source data with the PI server in Recovery mode (using only parameters **`/rst`** and **`/ret`**).

The **`Delete`** mode must be used only in conjunction with Recovery mode parameters (**`/rst`** and **`/ret`**) and can be enabled by adding the parameter (**`/mode=delete`**) to command line parameters.

Consider the example and figure provided in the **Data Recovery section**. If it is required to delete data from the PI server contained in time frame [12/15/2007 16:00:00 – 05/11/2008 2:00:05], then sufficient command line parameters should be:

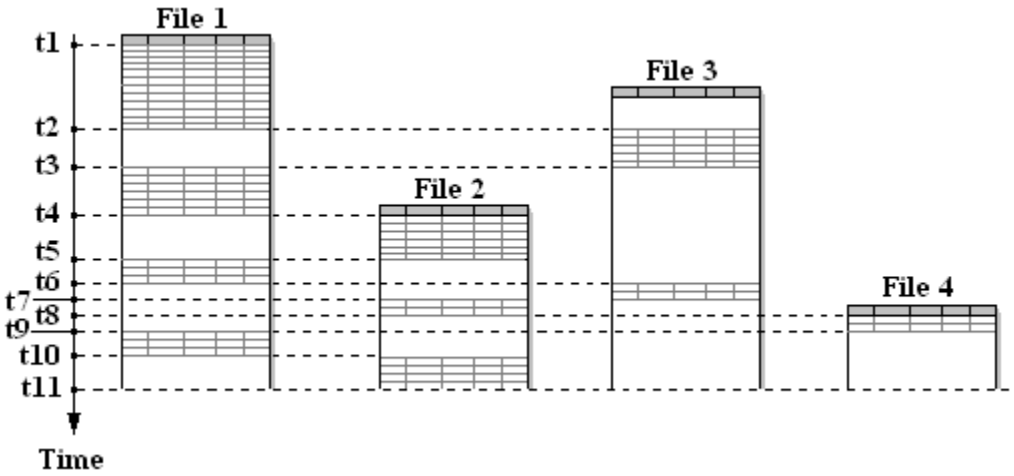
```
/rst="12/15/2007 16:00:00" /ret="05/11/2008 2:00:05" /mode=delete
```

If you need to delete all batches from specific time in the past until current time, the command line parameters should be:

```
/rst="12/15/2007 16:00:00" /mode=delete
```

EVT Source – Event Based Time Ordered Processing

The Batch interface processes EVT files based on the timestamps of each row within each EVT file, rather than processing EVT files based on names. On each scan, the interface performs a preliminary EVT directory scan to create a time ordered processing queue of all active EVT files based on the current position's timestamp within each file. It then scans each EVT file for the end position. This strategy allows creating a fixed time frame common to all EVT files. The interface will then read data, in the time frame, in time order. Any processing delays due to network losses, server unavailability, slow scan rates, and non alphabetical file naming can be handled gracefully by this approach. The following figure illustrates the difference in alphabetical file name versus event time ordered processing.



There are four EVT files starting at different times and containing data written sequentially by Batch Execution System. The vertical axis is the file time, where the **t1** is the earliest time and the **t11** is the latest time. The alphabetical file name processing sequence is given in the table below.

Order	File Name	Data Segments [start time – end time] written by the BES in parallel EVT files									
		t1 – t2	t2 – t3	t3 – t4	t4 – t5	t5 – t6	t6 – t7	t7 – t8	t8 – t9	t9 – t10	t10 – t11
1	File 1	X		X		X				X	
2	File 2				X			X			X
3	File 3		X				X				
4	File 4								X		

The event time ordered processing is illustrated in the table below

Order	File Name	Data Segments [start time – end time] written by the BES in parallel EVT files									
		t1 – t2	t2 – t3	t3 – t4	t4 – t5	t5 – t6	t6 – t7	t7 – t8	t8 – t9	t9 – t10	t10 – t11
1	File 1	X									
2	File 3		X								
3	File 1			X							
4	File 2				X						
5	File 1					X					
6	File 3						X				
7	File 2							X			
8	File 4								X		
9	File 1									X	
10	File 2										X

Dealing with Irrelevant Recipes

Sometimes it is necessary to exclude particular recipes from being processed into PI Server. Such recipes can be excluded from processing by using the INI file command **`skiprecipes`**. The switch contains the list of all recipes or recipe masks the interface should not evaluate. Everything related to specified recipe is not processed into PI Batch Database, PI Module database and PI Points databases. The filtering supports the following recipe types: Procedure, UnitProcedure, Operation, and Phase. For each event to be processed, the interface will check for the match in the recipe fields depending on recipe type:

Procedure Recipe : [Procedure] field of event

Unitprocedure Recipe: [UnitProcedure] field of event

Operation Recipe: [Operation] field of the event

Phase Recipe: [Phase] field of the event

Multiple recipes or their masks can be specified with a comma separator. If there is a space in the recipe name, use double quotes for the entire switch (not required if specified in INI file).

The following table represents available wildcards which can be specified within the recipe name mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Example(ini file): skiprecipes=recipe1,prc_*nt2 or **skiprecipes=recipe 1, prc_paint 2**

Dealing with Irrelevant Units

It is sometimes possible to use "virtual" or "dummy" units in a recipe that do not exist physically but aid in control transfer between recipes. In such cases, there could be overlapping PIUnitBatches on these "dummy" units which could lead to incorrect PIUnitBatch end times. These units can be excluded from processing the file by using the INI file command **skipunits**. The switch contains the list of all units the interface should not evaluate. Everything related to this unit is not processed into PI Batch Database or PI Points. The interface looks for the value in the [UNIT] field of each event. If any of those values match any of the units in the INI file **skipunits** list, the interface will simply move to the next event to be processed. The unit name comparison is not case sensitive and the interface does not compare the [AREA] or [ProcCell] values. Multiple unit names can be specified with a comma separator. If there is a space in the unit name, use double quotes for the entire switch. The unit masks can be specified as valid units. The following table represents available wildcards which can be specified within the unit name mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Example (ini file): skipunits=unit1,u*t2 or **skipunits=unit 1, unit 2**

Dealing with Irrelevant Phases

It is sometimes possible to use "virtual" or "dummy" phases or some phases are of no interest. These phases can be excluded from processing the file by using the INI file command **skipphases**. The parameter contains the list of all phases the interface should not evaluate. Everything related to this phase is not processed into PI Batch Database or PI Points. The interface looks for the value in the [Phase] or [PhaseModule] fields in each event. If any of those values match any of the phases in the INI file **skipphases** list, the interface will simply move to the next event to be processed. The phase name comparison is not case sensitive and instance number independent. Multiple phase names can be specified with a comma separator. If there is a space in the phase name, use double quotes for the entire switch. Phase masks can be specified as valid phases. The following table represents available wildcards which can be specified within the phase name mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Example (ini file): **`skipphases=phase_1,ph*2`** or **`skipphases=phase_1, ph*2`**

Dealing with Irrelevant Phase States

By default, the Batch interface creates every Phase State from the source (except states preceding RUNNING state). Sometimes it is desired to exclude some Phase States from being reported in PI server. The can be achieved by specifying the INI file command **`excludestates`** which defines the list of Phase States to be ignored. For each event to be processed, the interface checks for the match in the Phase State field (batch recipe). If match is found, the interface will mark particular Phase State as not visible and will not write it into PI server. Internally the interface still creates every Phase State to properly set Start and End times for all Phase States, visible and not. The name comparison is not case sensitive and allows masks as valid Phase State identifiers. Multiple phase states can be specified with a comma separator.

The following table represents available wildcards which can be specified within the phase state name mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Example (ini file): **`excludestates = COMPLETE , ABO*NG`**

Initialization File

The **Initialization** file: **`PIEMDVB<interfaceid>.ini`** is used to specify the interface configurations, such as data sources, translations, product template, equipment template, tag templates and property templates.

The command line syntax:

`/rst="12/05/2008 12:05:23" /merge`

Equivalent Initialization file defined parameters:

`rst=12/05/2008 12:05:23`

`merge = true`

> **Note:** In the initialization file every single parameter should be defined on separate line. There should be only one equal (**=**) sign per line. Specify two forward slashes (**//**) to comment any line in the INI file.

//rst=12/05/2008 12:05:23

//merge = true

In this case rst and mode parameters are commented, therefore they are considered to be undefined.

The initialization can contain any free text. The lines which will be attempted to be loaded by the interface are the lines with embedded equal sign (**=**).

Sample Initialization file – **`PIEMDVB.ini:`**

```
[Source Template]
Source[1].evtdir=”c:\test\evt”
Source[2].evtdir=\\deltav9\\journals\evt

[General]
Excludestates=COMPLETE, ABORTING
Equipment = abs:[Unit]\[PhaseModule]\Misc
[Tag Template]
// [Basic Tag template, triggered on Event=Report, aliases are
created as tag name]
Tag[6].Name  = [Unit]_[PhaseModule]_Report
Tag[6].Value = [Pval]
Tag[6].Type  = float

// [Tag template with custom aliases, triggered on Event=Owner
Change]
Tag[7].Name  = [Unit]_[PhaseModule]_Owner Change
Tag[7].Value = [time]_[Descript]
Tag[7].Type  = string
Tag[7].unitalias = [PhaseModule] Owner Change Me
Tag[7].phasealias = Owner Change Me

// [Tag template with custom aliases, triggered on set of events
defined as triggers]
// [Note: Unitalias and Phasealias are NOT going to be created since
there are no Unit or Phase
// Module defined in the tag name]
Tag[8].Name = Generic Tag
Tag[8].Value = [time]_[Event]_[BatchID]_[pval]
Tag[8].Type = string
Tag[8].trigger = Report
Tag[8].trigger = Owner Change
Tag[8].trigger = Operator Prompt
Tag[8].unitalias = [phasemodule] abcd
Tag[8].phasealias = testing

[Property Template]
Property[3].Value = [Time] State Change  [Descript] [pval]
// Property[20].Value = [Time][Pval][Event]  ← Disabled Property
template.
```

EVT Source – Example Event File Journal

An annotated journal file is shown in the table below. For clarity, the entire journal file has not been shown. Those fields that have been excluded do not significantly affect the logic used by the interface to process the event journal. The last column shown in the table below describes the actions taken by the interface in response to each message listed in the journal file.

Abbreviated Example EVT file demonstrating interface actions pertaining to PIModule and PIBatch object creation

LclTime	Descript	Event	Pvalue	EU	Action Taken by Interface
2000.05.17 14:06:17		Event File Name	\\OBATCH\JOURNALS\15.evt		New event file detected.
2000.05.17 14:06:17	Author	Recipe Header	PR1Upauthor		Interface waiting for "Beginning of Batch."
2000.05.17 14:06:17	Product Description	Recipe Header	PR1UPProcedureDescription		
2000.05.17 14:06:17	PR1UPPROCDESCRIPT	System Message	Beginning Of BATCH	Procedure	PIBatch object created. Start time assigned to PIBatch.
2000.05.17 14:06:30	Procedure Started.	System Message	0		
2000.05.17 14:06:30	Step Activated	Step Activity	UP2OPS:1	Unit Procedure	PIUnitBatch created to represent the unit procedure.
2000.05.17 14:06:30	Unit Procedure Started	System Message	0		
2000.05.17 14:06:30	Unit Acquired	Arbitration	SOL_DELIV_1		Start time assigned to PIUnitBatch.
2000.05.17 14:06:30	Step Activated	Step Activity	OP1PHASE:1	Operation	PISubBatch created to represent the operation.
2000.05.17 14:06:30	Operation Started	System Message	0		Start time assigned to PISubBatch.
2000.05.17 14:06:30	Equip. Module Acquired	Arbitration	S1_TEMPERATURE1		
2000.05.17 14:06:30	Step Activated	Step Activity	SX_TEMPERATURE:1	Phase	Sub-PISubBatch created to represent the phase.
2000.05.17 14:06:32	State Changed:	State Change	RUNNING		Start time assigned to Sub-PISubBatch.
2000.05.17 14:07:06	State Changed:	State Change	COMPLETE		End time assigned to the Sub-PISubBatch.
2000.05.17 14:07:07	Equip. Module Released	Arbitration	S1_TEMPERATURE1		
2000.05.17 14:07:07	Operation Finished	System Message	0		End time assigned to the PISubBatch.
2000.05.17 14:07:45	Unit Released	Arbitration	SOL_DELIV_1		End time assigned to the PIUnitBatch.

LclTime	Descript	Event	Pvalue	EU	Action Taken by Interface
2000.05.17 14:07:45	Unit Procedure Finished	System Message	0		
2000.05.17 14:09:05	Procedure Finished.	System Message	0		
2000.05.17 14:09:05	PR1UPPROCDESCRIPT	System Message	End Of BATCH	Procedure	End time assigned to the PIBatch.

Chapter 3. Installation Checklist

If you are familiar with running PI data collection interface programs, this checklist helps you get the Interface running. If you are not familiar with PI interfaces, return to this section after reading the rest of the manual in detail.

This checklist summarizes the steps for installing this Interface. You need not perform a given task if you have already done so as part of the installation of another interface. For example, you only have to configure one instance of Buffering for every interface that runs on an Interface Node.

The Data Collection Steps below are required. Interface Diagnostics are optional.

Data Collection Steps

1. Confirm that you can use PI SMT to configure the PI Server. You need not run PI SMT on the same computer on which you run this Interface.
2. If you are running the Interface on an Interface Node, edit the PI Server's Trust Table to allow the Interface to write data.
3. Run the installation kit for this Interface. This kit runs the PI SDK installation kit, which installs both the PI API and the PI SDK. This kit also runs the PI Event Frames Interface Manager installation kit, which installs a configuration tool for the interface.
4. If you are running the Interface on an Interface Node, check the computer's time zone properties. An improper time zone configuration can cause the PI Server to reject the data that this Interface writes.
5. Run the PI Event Frames Interface Manager and configure a new instance of this Interface. Essential startup parameters for this Interface are

   Point Source (`/PS=c`)
   Interface ID (`/ID=#`)
   PI Server (`/Host=node:port`)

   Define sources

6. `Location1` is the interface instance.
   `Location2` is the index of the tag.
   `Location3` is the point type
   `Location4` is the scan class
   `ExDesc` contains the copy of tag name created by the interface.
   `InstrumentTag` is the unit.
7. Start the Interface interactively with command line parameter **`/mode=stat`** (this mode allows only read from data sources and PI server) and confirm its successful connection to the PI Server and data sources.

Note, this interface does not use the PI API, therefore PI Buffering (pibufss/bufserv) is not required by the interface.

8. Confirm that the Interface collects data successfully.
9. Configure the Interface to run as a Service. Make sure that interface is **NOT** set as PI Buffer dependent. Confirm that the Interface runs properly as a Service.
10. Restart the Interface Node and confirm that the Interface restarts.

Interface Diagnostics

The PI Interface to the Emerson DeltaV Batch can be diagnosed through the use of performance points automatically created by each instance of the interface. There are 35 performance tags which are broken into 3 categories:

- Health Monitoring
- Object counters.
- Timers.

All performance tags are prefixed by the interface name and the interface ID as follows:

Prefix : PIEMDVB_<Interfaceid>

In the following paragraphs, Prefix is used to create actual performance tag names.

Health Monitoring

There are two tags designed to monitor the health of the interface: heartbeat tag and the device status tag. The heartbeat tag is updated with the frequency defined by the **`/scan`** parameter in command line parameters. If scan time is greater than 60 seconds than the heartbeat tag is updated every 60 seconds. The heartbeat tag value is defined as a cycle of integer numbers from 1 to 15. The device status tag is automatically configured and created if missing by the interface on startup. The following events can be written into the device tag:

a) "Good" – the interface is properly communicating and reading data from the data sources.

b) "1 | Starting" – the interface goes through the initialization routines.

c) The following events represent proper communication with the data sources. This message is displayed on successful connection to each source.

"2 | Connected/No Data | EVT Directory Monitor: <directory name> Initialized."

"2 | Connected/No Data | Source SQL Server: <server name> Initialized."

d) The following list of events represents the failure to communicate with the Event Journal file directory, or failure to read data from the Event Journal File:

"3 | 1 device(s) in error | Error monitoring directory (onError): <directory name>"

"3 | 1 device(s) in error | Error monitoring directory: <directory name>"

"3 | 1 device(s) in error | Failed to start directory monitoring thread: <directory name>"

"3 | 1 device(s) in error | Error in scanning directory: <directory name>"

"3 | 1 device(s) in error | Error obtaining EVT files EOF."

"3 | 1 device(s) in error | Error getting current EVT file timestamp."

"3 | 1 device(s) in error | Error reading EVT file: <filename>."

"3 | 1 device(s) in error | Error while reading EVT file."

"3 | 1 device(s) in error | Error reading SQL Server: <server name>."

The properties of Health monitoring tags are provided in the table below, where the **Prefix** represents `PIEMDVB_<Interfaceid>`. Note, these tags are automatically created by the interface if they are not found in the PI server upon startup.

Tag Name	Point Type	Loc1	Loc3	PointSource	ExcDesc
Prefix_DeviceStatus	Int32	Intf ID	0	Intf Pt Src	[UI_DEVSTAT]
Prefix_HeartBeat	Int32	Intf ID	1	Intf Pt Src	[UI_HEARTBEAT]

Object Counters

There are 24 tags designed to monitor performance of the interface based on the number of different type objects read from the source and written to the PI server. These tags are automatically created on first interface startup. Archiving flag for these tags is turned off. The following table contains common point attributes for this group of tags:

Point Type	Location1	Point Source
Int32	<Interface ID>	<Interface Point Source>

The specific attributes for each performance counter tag are provided in the table below where the Prefix is defined as `PIEMDVB _<Interfaceid>`. All Counters are reset on interface startup.

Tag Name	Loc 3	ExcDesc	Description
Prefix_EventReadCount	2	[UI_EVENTREADCOUNT]	Number of events read from the source since last startup.
Prefix_ErrorCount	3	[UI_ERRORCOUNT]	Number of errors occurred since last startup.
Prefix_SourceUnitCount	4	[UI_SOURCEUNITCOUNT]	Number of Units found on the data source(s) since startup.
Prefix_PIUnitCount	5	[UI_PIUNITCOUNT]	Number of Units found and added on the PI server since startup.
Prefix_SourcePhaseModCount	6	[UI_SOURCEPHASEMODCOUNT]	Number of Phase Module found on the data source(s) since startup.
Prefix_PIPhaseModCount	7	[UI_PIPHASEMODCOUNT]	Number of Phase Module found and added on the PI Server since startup.

Tag Name	Loc 3	ExcDesc	Description
Prefix_SourceBatchCount	8	[UI_SOURCEBATCHCOUNT]	Number of batches found on the data source(s) since startup.
Prefix_PIBatchCount	9	[UI_PIBATCHCOUNT]	Number of PIBatch objects found and added on the PI server since startup.
Prefix_SourceUnitBatchCount	10	[UI_SOURCEUNITBATCHCOUNT]	Number of unitbatches found on the data sources(s) since startup.
Prefix_PIUnitBatchCount	11	[UI_PIUNITBATCHCOUNT]	Number of PIUnitBatch objects found and added on the PI server since startup.
Prefix_SourceSubBatchCount	12	[UI_SOURCESUBBATCHCOUNT]	Total number of operations+phases+phase states found on the data source since startup.
Prefix_PISubBatchCount	13	[UI_PISUBBATCHCOUNT]	Total number of PISubBatch objects founded and added to the PI server since last startup.
Prefix_SourcePropertyNodeCount	14	[UI_SOURCEPROPNODECOUNT]	Number of property nodes found in data source(s) since last startup
Prefix_PIPropertyNodeCount	15	[UI_PIPROPNODECOUNT]	Number of PIProperty objects (nodes) found and added to the PI server since last startup.
Prefix_SourcePropertyEventCount	16	[UI_SOURCEPROPEVENTCOUNT]	Number of events to be written to the batch properties found on the data source(s) since last startup.
Prefix_PIPropertyEventCount	17	[UI_PIPROPEVENTCOUNT]	Number of PIProperties(events) found and added to the PI server since last startup.
Prefix_SourceTagCount	18	[UI_SOURCETAGCOUNT]	Number of tags found on the data source(s) since last startup
Prefix_PITagCount	19	[UI_PITAGCOUNT]	Number of PIPoints found and added to the PI server since last startup.
Prefix_SourceTagEventCount	20	[UI_SOURCETAGEVENTCOUNT]	Number of events to be written into tags found on the data sources(s) since last startup.

Tag Name	Loc 3	ExcDesc	Description
Prefix_PITagEventCount	21	[UI_PITAGEVENTCOUNT]	Number of events written into PIPoints on the PI server since last startup.
Prefix_SourceTagAliasCount	22	[UI_SOURCETAGALIASCOUNT]	Number of tag aliases to be created based on the data source(s) since last startup.
Prefix_PITagAliasCount	23	[UI_PITAGALIASCOUNT]	Number of PIAliases found and added to the PI server since last startup.
Prefix_CachedBatchCount	24	[UI_CACHEDBATCHCOUNT]	Number of batch objects cached in the local memory.
Prefix_OpenBatchCount	25	[UI_OPENBATCHCOUNT]	Subset of cached objects which still have no end time set.
Prefix_WaitingForEquipmentUB	34	[UI_UBWAITFOREQUIP]	Number of UnitBatches which do not have equipment allocated yet. The allocation is check at PI Server synchronization routine.

Timers

The last performance tag category is composed of timer tags which are build automatically on first interface startup. Each timer tag reports on how much time per scan it took the interface to perform particular task. There are 3 task subcategories: data source reading, local data caching and synchronizing cached data with PI server. The following table provides common tag attributes for these tags:

Point Type	Location1	Point Source
Float32	<Interface ID>	<Interface Point Source>

The specific attributes for each performance timer tag are provided in the table below where the Prefix is defined as **PIEMDVB_<Interfaceid>.**

Tag Name	Loc3	ExcDesc	Description
Prefix_SourceReadTime	26	[UI_SOURCEREADTIME]	The time per scan it took the interface to read data from data source(s).
Prefix_TagCacheTime	27	[UI_TAGCACHETIME]	The time per scan it took the interface to populate local tag cache.
Prefix_BatchCacheTime	28	[UI_BATCHCACHETIME]	The time per scan it took the interface to populate the local batch cache.
Prefix_EquipmentCacheTime	29	[UI_EQUIPCACHETIME]	The time per scan it took the interface to populate the local equipment (module) cache.
Prefix_BatchSyncTime	30	[UI_BATCHSYNCTIME]	The time per scan it took

Tag Name	Loc3	ExcDesc	Description
			the interface to synchronize local batch cache with the PI server.
Prefix_TagSyncTime	31	[UI_TAGSYNCTIME]	The time per scan it took the interface to synchronize local tag cache with the PI server.
Prefix_EquipmentSyncTime	32	[UI_EQUIPSYNCTIME]	The time per scan it took the interface to synchronize local equipment cache with the PI server.
Prefix_TotalTime	33	[UI_TOTALTIME]	The total time per scan it took the 112atched112s to read data, cache it in the local memory and synchronize local cache wit PI server.

Chapter 4. Interface Installation

OSIsoft recommends that interfaces be installed on PI Interface Nodes instead of directly on the PI Server node. A PI Interface Node is any node other than the PI Server node where the PI Software Development Kit (PI SDK) has been installed (see the PI SDK manual). With this approach, the PI Server need not compete with interfaces for the machine's resources. The primary function of the PI Server is to archive data and to service clients that request data.

> **Note:** Buffering is not recommended with the PI Interface to the Emerson DeltaV Batch. This is due to the fact that the source data is already effectively buffered on the source.

In most cases, interfaces on PI SDK nodes should be installed as automatic services. Services keep running after the user logs off. Automatic services automatically restart when the computer is restarted, which is useful in the event of a power failure.

The guidelines are different if an interface is installed on the PI Server node. In this case, the typical procedure is to install the PI Server as an automatic service and interfaces as manual services that are launched by site-specific command files when the PI Server is started. Interfaces that are started as manual services are also stopped in conjunction with the PI Server by site-specific command files. This typical scenario assumes that Bufserv is not enabled on the PI Server node. Bufserv can be enabled on the PI Server node so that interfaces on the PI Server node do not need to be started and stopped in conjunction with PI, but it is not standard practice to enable buffering on the PI Server node.

Naming Conventions and Requirements

In the installation procedure below, it is assumed that the name of the interface executable is **`PIEMDVB.exe`**, the startup command file is called **`PIEMDVB.bat`**, and the initialization file is called **`PIEMDVB<interfaceid>.ini`**.

When Configuring the Interface Manually

When configuring the interface manually it is customary for the user to rename the executable, the startup command and initialization files when multiple copies of the interface are run. For example, **`PIEMDVB1.exe`**, **`PIEMDVB1.bat`** and **`PIEMDVB1.ini`** would typically be used for interface number 1, **`PIEMDVB2.exe`**, **`PIEMDVB2.bat`** and **`PIEMDVB2.ini`** for interface number 2, and so on. When an interface is run as a service, the executable and the command file must have the same root name because the service looks for its command-line parameters in a file that has the same root name.

Interface Directories

PIHOME Directory Tree

32-bit Interfaces

The `[PIHOME]` directory tree is defined by the `PIHOME` entry in the `pipc.ini` configuration file. This `pipc.ini` file is an ASCII text file, which is located in the `%windir%` directory.

For 32-bit operating systems a typical **`pipc.ini`** file contains the following lines:

```
[PIPC]
PIHOME=C:\Program Files\PIPC
```

For 64-bit operating systems a typical **`pipc.ini`** file contains the following lines:

```
[PIPC]
PIHOME=C:\Program Files (X86)\PIPC
```

The above lines define the root of the `PIHOME` directory on the C: drive. The `PIHOME` directory does not need to be on the C: drive. OSIsoft recommends using the paths shown above as the root `PIHOME` directory name.

Interface Installation Directory

The interface install kit will automatically install the interface to:

`PIHOME\Interfaces\EMDVB\`

`PIHOME` is defined in the **`pipc.ini`** file.

Interface Installation Procedure

The Batch interface setup program uses the services of the Microsoft Windows Installer. Windows Installer is a standard part of Windows 2000. To install, run the appropriate installation kit

`EMDVB`_#.#.#.#.exe

Installing the Interface as a Windows Service

The Batch interface service can be created with the PI Interface Configuration Utility, or can be created manually.

Installing the Interface Service with the PI Event Frames Interface Manager

The PI Event Frames Interface Manager provides a user interface for creating, editing, and deleting the interface service on the **Service** tab.

Service Tab

Settings for installing and running the interface as a Windows Service.

Setting	Description
Display Name	The name of service as displayed in the Services control Panel. To indicate that the service is part of the OSIsoft suite of products, prefix the name with “PI”.
Logon as	The Windows user account used to run the interface service.
Password	Password for the Windows user account used to run the interface service.
Startup Type	Configures whether the service starts automatically when the interface node is rebooted.
Dependencies	Configures other services that PIEMDVB interface requires in order to run.

Installing the Interface Service Manually

Help for installing the interface as a service is available at any time with the command:
`PIEMDVB.exe -help`

Change to the directory where the **`PIEMDVB1.exe`** executable is located. Then, consult the following table to determine the appropriate service installation command.

Windows Service Installation Commands on a PI Interface Node or a PI Server Node without Bufserv implemented	
Manual service	PIEMDVB.exe –install –depend tcpip
Automatic service	PIEMDVB.exe –install –auto –depend tcpip
*Automatic service with service id	PIEMDVB.exe –serviceid X –install –auto –depend tcpip

*When specifying service id, the user must include an id number. It is suggested that this number correspond to the interface id (/id) parameter found in the interface .bat file.

Check the Microsoft Windows services control panel to verify that the service was added successfully. The services control panel can be used at any time to change the interface from an automatic service to a manual service or vice versa.

Chapter 5. Digital States

For more information regarding Digital States, refer to the PI Server documentation.

Digital State Sets

PI digital states are discrete values represented by strings. These strings are organized in PI as digital state sets. Each digital state set is a user-defined list of strings, enumerated from 0 to n to represent different values of discrete data. For more information about PI digital tags and editing digital state sets, see the PI Server manuals.

An interface point that contains discrete data can be stored in PI as a digital tag. A Digital tag associates discrete data with a digital state set, as specified by the user.

System Digital State Set

Similar to digital state sets is the system digital state set. This set is used for all tags, regardless of type to indicate the state of a tag at a particular time. For example, if the interface receives bad data from an interface point, it writes the system digital state `bad input` to PI instead of a value. The system digital state set has many unused states that can be used by the interface and other PI clients. Digital States 193-320 are reserved for OSIsoft applications.

Chapter 6. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For example, the string *EV* may be used to identify points that belong to the Batch Interface. To implement this, the PointSource attribute would be set to *EV* for every PI Point that is configured for the Batch Interface. Then, if **/ps=EV** is used on the startup command-line of the Batch Interface, the Interface will search the PI Point Database upon startup for every PI point that is configured with a PointSource of *EV*. Before an interface loads a point, the interface usually performs further checks by examining additional PI point attributes to determine whether a particular point is valid for the interface. For additional information, see the **/ps** parameter.

Case-sensitivity for PointSource Attributes

The PointSource character that is supplied with the **/ps** command-line parameter is not case sensitive. That is, **/ps=P** and **/ps=p** are equivalent.

Reserved Point Sources

Several subsystems and applications that ship with the PI is associated with default PointSource characters. The Totalizer Subsystem uses the PointSource character **T**, the Alarm Subsystem uses **G** and **@**, Random uses `R`, RampSoak uses `9`, and the Performance Equations Subsystem uses **C**. Do not use these PointSource characters or change the default point source characters for these applications. Also, if a PointSource character is not explicitly defined when creating a PI point; the point is assigned a default PointSource character of `Lab` (PI 3). Therefore, it would be confusing to use `Lab` as the PointSource character for an interface.

> **Note:** Do not use a point source character that is already associated with another interface program. However it is acceptable to use the same point source for multiple instances of an interface.

Chapter 7. PI Point Configuration

The PI point is the basic building block for controlling data flow to and from the PI Server. The batch interface automatically builds all points based on the information found in INI file.

Interface-specific Points

Process parameters are often specified in batch data sources. These parameters are typically more easily viewed as a graphical trend. Points may be built to specify which events are to be captured and stored in PI. Please refer to section **Event Logging – Tag Template** for information on how to configure Tag Templates for specific event capturing.

Chapter 8. Startup Command File

Command-line parameters can begin with a **/** or with a **-**. For example, the **/ps=E** and **-ps=E** command-line parameters are equivalent.

For Windows, command file names have a .bat extension. The Windows continuation character (**^**) allows for the use of multiple lines for the startup command. The maximum length of each line is 1024 characters (1 kilobyte). The number of parameters is unlimited, and the maximum length of each parameter is 1024 characters.

Configuring the Interface with PI Event Frames Interface Manager

The PI Event Frames Interface Manager provides a graphical user interface for configuring the PI Interface to the Emerson DeltaV Batch. If the interface is configured with this tool, the batch file of the interface (PIEMDVB**.bat**) and the interface settings file (PIEMDVB.ini) will be created and maintained by the PI Event Frames Interface Manager and all configuration changes will be kept in that file. The procedure below describes configuration using PI Event Frames Interface Managerto configure the Batch Interface.

To configure PI Event Frames Generator (PIEFGen), you use the PI Event Frames Interface Manager configuration tool. The following sections describe the settings required to configure event frame generation. The command line parameters that are generated in the interface batch startup file are listed where appropriate, to help you verify that you have created the desired configuration.

Interface Selection Tab

To create a new instance of PIEMDVB, perform the following steps:

1. Click **Add Interface**. A browse dialog is displayed.
2. Browse to the `%PIPC%\Interfaces\PIEMDVB` directory, select `PIEMDVB.exe`, and click **Open** to dismiss the browse dialog.
3. Click **OK**. The **Interface** field displays the name of the PIEMDVB instance you created.

The PI Event Frames Interface Manager can be used to manage multiple instances of PIEMDVB.

File Selection Tab

The **File Selection** tab is used to select the interface settings file that stores settings and the configuration for the interface instance:

- Interface Settings File (.INI): Contains the interface startup parameter and configuration settings. Be sure to specify the .ini extension.

Server Information Tab

The **Server Information Tab** is where you specify the PI Server and PI AF Server systems that you intend to use with the interface instance. PIEMDVB stores data in PI Tags on a PI Server. It can generate either batches in the PI Server Batch Database or event frames on a PI AF Server.

PI Server (/HOST)

Specifies the PI Server node to which PIEMDVB writes PI tag data. **Host** is the IP address of the PI Sever node or the fully qualified domain name of the PI Server node. If the PI Server you want to use is not in the drop down list, you must add it to the known servers table using the **AboutPI-SDK** application.

[PI Server] User and [PI Server] Password

For PI Servers version 3.4.380.36 and higher, use Windows Integrated Security for authentication. Omit the user name and password from these fields, and ensure that the Windows account that runs the interface has sufficient permissions on the PI Server to write data to PI Points.

For PI Servers prior to version 3.4.380.36, configure a trust on the PI Server that permits accesss for the user running the interface or interface service.

[PI Server] Port

The port number for TCP/IP communication. The default port (recommended) is 5450.

Use PI AF server

Check this box to create event frames on a PI AF server, instead of creating batches in the PI Server Batch Database.

[PI AF] Host and Database (/AFHOST and /AFDATABASE)

The destination PI AF server node and database where you want PIEMDVB to create event frames.

[PI AF] User and [PI AF] Password

If you are not using Windows Integrated Security for authentication (recommended), enter the user name and password for the Windows user account that you intend to use to connect to PI AF.

Source

On this tab, you define the data sources from which the PIEMDVB interface will read data. The interface can read data from multiple data sources, but each interface instance can only read from one type of data source. To read from multiple different kinds of data sources, such as EVT file or SQL server, create multiple interface instances.

To read from EVT file: Add an EVT source, and specify the directory where the EVT files are.

To read from SQL server: Add a SQL server and configure it.

SQL Server

Specifies the SQL server machine name or computer name.

SQL Database

Specifies the database to read data from.

User and Password

The user account and password the interface uses to connect to the SQL server.

Use Alarms and Events Historian

Check this box to read Alarms and Events data from the SQL server. You will then have additional options to specify the Area to Process Cell or the Unit to Process Cell mapping, or to specify the DeltaV equipment hierarchy file that specifies these mappings.

DeltaV OPCAE OPC Node and OPC Server

Specify that Alarms and Events will be collected from the source OPC machine node name and OPC server.

Filters Tab

This tab configures the Recipes, Units, Phases, or Phase States that should be excluded from processing by the Batch Interface.

Skip Phases (/SKIPPHASES)

The interface will not process any event with the listed phases in the [Phase] or [Phasemodule] column.

Skip Units (/SKIPUNITS)

The interface will not process any event with the listed units in the [Unit] column.

Skip Recipes (/SKIPRECIPES)

The interface will not process any event with the listed recipes in the appropriate column ([Procedure] for a procedure recipe, [UnitProcedure] for a unitprocedure recipe, etc.)

Exclude Phase States (/EXCLUDESTATES)

The interface will not write phase state events to the PI system with the listed phase state in the [PhaseState] column.

Time Settings Tab

This tab configures the time settings that control how PIEMDVB handles server connections and processes data.

Query Time Settings

Scan (/SCAN=<seconds>)

Specifies how frequently the interface scans the data source.

Cache time (/CACHETIME=<days>)

Specifies how long completed events are retained in memory. Default is one day. Specify the maximum duration expected between event frames that need to be merged, plus any desired margin of safety.

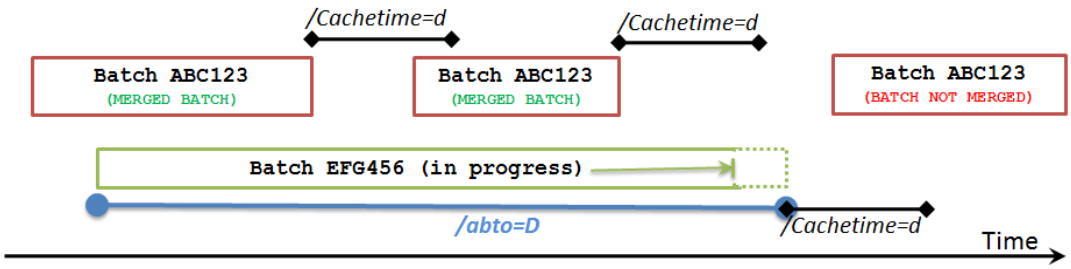
The value can be specified as whole day or fraction of a day. For example, to release completed batches when their end time is less than 7 days and 12 hours from current time, specify the following cache time setting:

```
CACHETIME=7.5 days
```

Abandoned batch timeout (/ABTO=<days>)

Specifies how long event frames can remain open before being considered abandoned. When this period (plus cache time) elapses, the interface writes an end time to the event to close it. Specify the maximum duration expected for events, plus any desired margin of safety.

For example, if you set abandoned batch timeout to 50.5 days and cache time is seven days, events open for 57.5 days are automatically closed. The following figure illustrates timeout logic.



Maximum query time frame (/MAXQTF=<days>)

To help manage workload and memory usage, defines the maximum time frame for queries. For example, if you specify 30 days and the interface queries for one year's worth of data, the interface issues 12 one-month queries rather than one (large) one-year query.

Maximum stop time (/MAXSTOPTIME=<seconds>)

Specifies the maximum time allowed for the interface to properly shutdown. If the shutdown process takes longer than the specified time, the interface is forced to terminate immediately. The default value is 120 seconds.

Use local time stamps to process incoming events (/TS)

Applies the time on the local machine to events from the datasource. By default, timestamps are recorded using GMT.

PI Connection Settings

PI connection timeout (/PICONNTO=<seconds>)

Override the default PI SDK Connection TimeOut property.

PI data access timeout (/PIDATO=<seconds>)

Override the default PI SDK Data Access TimeOut property.

Retry (/RETRY=<seconds>)

Specifies how long to wait before retrying a failed SDK attempt to write data to PI Server. The default is 60 seconds.

Retry timeout (/RETRYTO=<seconds>)

Specifies timeout for failed SDK attempts to write data to PI Server. To avoid data loss, set to 0 (default, no timeout).

SQL server Connection Settings

SQL connection timeout (/SQLCONNTO=<seconds>)

Override the default SQL connection timeout. The default is 60 seconds.

SQL data access timeout (/SQLDATO=<seconds>)

Override the default SQL data access timeout. The default is 100 seconds.

Operational Settings Tab

The settings on this tab configure the mode in which PIEMDVB runs and other, related settings.

Runtime mode (/MODE=<mode>)

Interface modes are as follows:

Mode	/MODE Flag	Description
Realtime (default)	REALTIME	Scan data source to collect data in realtime
Recovery	REALTIME with /RST specified	Scan data source and generate or correct events accordingly. PIEMDVB always starts in recovery mode, then switches to realtime mode.
Statistics	STAT	Compare data source history against events and report results without updating any data.
Delete	DELETE with /RST and /RET specified	Delete events for a specified period.

Perform one scan then stop (/SINGLERUN)

The interface performs one scan of active points, then exits.

Print result of first scan to file (/PRINT=<file name>)

The name of the text file to which the results of the first scan are printed. The results include the event frame hierarchy tree, the tag list, and the equipment tree. This parameter is designed primarily for troubleshooting and configuration testing when the interface is run in statistics mode.

Debug level (/DB=<#>)

Specifies level of detail for logging as follows:

- 0: Log errors and warnings (default)
- 1: Log errors, warnings and major successes
- 2: Verbose logging

Numeric settings (/NS=<lang>)

Configures how numeric values are formatted by the interface, to enable the interface to properly interpret numeric values based on the machine's regional setting or a user-specified language. Default is "English_UnitedStates".

Interface ID (/ID=x)

Specifies the numeric interface instance identifier (maximum nine digits). To detect PI points maintained by the interface instance, the interface matches this setting against the value in the points' `Location1` attribute.

Point source (/PS=x)

Point source for the interface instance. Point source is not case sensitive. Corresponds to the `PointSource` attribute of individual PI Points. The interface loads PI points with the same point source. The default point source is "PIEMDVB".

Associate all reference elements with child Event Frames (/DPRETC)

When creating Event Frames in PI AF, by default all reference elements are associated with child event frames.

Failover Settings

Failover tag (/FAILOVERTAG=x)

The tag on the PI Server that will be used to coordinate failover.

Failover identifier (/FAILOVERID=x)

The unique identifier of this interface in failover.

Failover swap time (/SWAPTIME=<seconds>)

How long an interface should be inactive before another interface will assume writing data to the PI system.

Security Settings

Specify point security (/PTSEC=x)

Override the default ptsecurity of PIPoints created by the interface.

Specify data security (/DATASEC=x)

Override the default datasecurity of PIPoints created by the interface.

Save Settings Tab

Saves the configuration.

Test Configuration Tab

Tests the configuration settings. Specify test settings as follows, click **Run Test**, then check the output file for results.

Field	Description
`BAT File`	The path to the interface .BAT file to run during the test execution.
`Output File`	The path to the text file where the test results are written.
`Start Time`	Start time for scanning.
`End Time`	End time for scanning

Configuring Interface Startup Files

The interface has two startup configuration files; **PIEMDVB.bat** and **PIEMDVB.ini**. The .bat file is required and is the primary file for specifying interface configurations. The INI file is used to specify the interface configurations, such as data sources, translations, product template, equipment template, tag templates and property templates.

When using the .INI file, each parameter should be defined on separate line. There should be only one equal (=) sign per line. Specify two forward slashes (//) to comment any line in the INI file.

When configuring the .bat startup file the continuation character ^ can be used to allow multiple lines for defining parameters. The maximum length for a single line is 1024 characters (1 kilobyte). This is a Windows limitation.

Command-line Parameters

This is a listing of the command-line parameters and their specific behavior with respect to the PI Batch interface. This section gives more detailed information concerning the parameters that may be specified when configuring the interface (such as with the PI ICU).

Parameter	Description
/abto=<days> Optional Default: **100** days	(**A**)Bandoned (**B**)atch (T)ime(**O**)ut. Defines the time period from the cached batches time frame into the past after which the open batches are considered to be abandoned and can be released from the interface's local cache. The default value is 100 days. Example: If **/abto=50.5** and **/cachetime=7.1** then the batches with last event occurred before NOW() – 7.1 days – 50.5 days will be considered abandoned and removed from the local interface memory. --|----------[-cached batches time frame -] -→ Timeline -57.6 days -7.1 days (current time)
/adu=true/false Optional Default: **true**	(A)llow (D)eferred (U)Units. It enables/disables support for deferred units. Commonly used when the unit allocation occurs at the Phase level of the recipe. When /adu=true, it allows creation. UnitBatch level objects with the same timestamps as at the source, even though the Unit field is empty and the interface cannot create the PIUnitBatch on PI Server yet.

/bidm=<list> Optional	The **/bidm** switch (**B**atch **ID** **M**ask) is used to obtain a new BatchID, which is a substring of the value in the source BatchID field. The **/bidm** takes a list of masks as the input value. Each BatchID mask can consist of an array of valid symbols and wildcards. The following wildcards are supported the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous BatchID mask symbol * - any array of ? symbols. Example: Let's say that the BatchID column in the event file is **lot30112** / **90dev123** / **12345stp / ld567**. The /bidm="#####" will result in new BatchID **30112**. The /bidm="##@!" will result in new BatchID **90dev**. The /bidm="*##@!" will result in new BatchID **lot30112** / **90dev**. The /bidm="@@@@, #8dev4, #!" will result in new BatchID **30112**. Since the first and second masks could not be found, third mask is used instead.
/cachetime=<days> Optional Default: **1** day	Defines the time period for which the completed batches are retained in the memory. [(*-cachetime) - *] The default value is 1.0 day. The value can be specified as whole day or fraction of the day. Example: /cachetime=7.5 days In this case the interface is going to release completed batches when their end time is going to be less than 7 days and 12 hours from current time.
/dac Optional	The /dac Disable Arbitration Counters parameter informs interface to release unit on the first Resource Release event even though the number of Acquire events is higher than number of Release events. By default, interface requires number of Resource Release events to be the same as Resource Acquire events for each unit to release the unit.
/damcae Optional	The /damcae Disable Auto Module Creation for A&E data source switch informs the interface to ignore events originated from A&E data source when creating/checking PI MDB objects. **Note**: if AlarmTag[#].Alias = <module path> | <alias name> <module path> contains $ (root node symbol), then defined module path is going to be checked/created regardless of /damcae switch. Example: AlarmTag[1].Alias= $\Test\[Module] | SomeAliasName In this example, the interface is going to create hierarchy in PI MDB starting from MDB root node, or leaf node of the start module path if /smp=<start module path)> is defined in command line.

/DataSec=<string> Optional	The **/DataSec** parameter specifies the PIPoint Data Access Security rights. These rights are assigned to interface-generated tags during point creation. This string has different forms. If PIPoints are created on a PI Server 3.4.375.99 or earlier, it will have an owner, group, world format. Example: /datasec="**o:rw g:r w:r**" If PIPoints are created on a PI Server 3.4.380.36 or later, it must specify an Access Control List (ACL) Example: /datasec="piadmin: A(r,w) | PIEngineers: A(r)"
/db=[#] Optional Default: **0**	The **/db=[#]** parameter specifies the Interface debug logging message level. There are three levels that may be assigned: 0 – Log only errors and warnings. 1 – Log errors, warnings and major success messages 2 – Log ALL messages. Log level two (2) is the most verbose setting; while level zero (0) reports the least detail (it logs only error messages). The default logging level is **0**, to log errors and warnings only. When testing the Interface, it may be necessary to use a more verbose setting (1 or 2).
/dpretc Optional AF Only	By default, the interface propagates each event frame element reference to its children event frames. This functionality can be disabled by specifying the following command line parameter: /dpretc - (D)isable (P)ropagation of (R)eferenced (E)lements (T)o (C)hilren.
/dumpread=<filename> Optional	This parameter allows to read data directly from the dump file. It should be used only for troubleshooting.
/dumpwrite=<filename> Optional	This parameter allows the interface to generate dump file with data currently being processed by the interface. It should be used only for troubleshooting.
/EquipmentXML=<file path> Optional	Specifies the location of the DeltaV-generated equipment hierarchy XML file. The EMDVB Interface uses this reference data to locate missing ProcessCell field by searching based on the combination of Area and Unit fields. Valid only when a DeltaV AE SQL datasource is defined. Example: /EquipmentXML="C:\DeltaV\EquipHier.xml"
/FailOverID=<string> Optional	Configure the unique failover ID for the interface instance. Must be used with the `/FailOverTag` parameter. Example: /FailOverID="intf1"

/FailOverTag=<PI Point Name> Optional	Specifies the PI point that is used to track which interface instance is primary. Must be used with the /FailOverID parameter. Example: /FailOverTag="Batch_FailoverTag"
/host=host:port Required	Specifies the PI server node where data is going to be stored. **Host** is the IP address of the PI Sever node or the domain name of the PI Server node. **Port** is the port number for TCP/IP communication. The port is always 5450. It is recommended to explicitly define the host and port on the command-line with the **/host** parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults. Examples: The interface is running on a PI Interface Node, the domain name of the PI home node is Marvin, and the IP address of Marvin is 206.79.198.30. Valid **/host** parameters would be: **/host=marvin** **/host=marvin:5450** **/host=206.79.198.30** **/host=206.79.198.30:5450**
/id=x Required	The **/id** parameter is used to specify the interface identifier. The interface identifier is a string that is no longer than 9 characters in length. This interface uses the **/id** parameter to identify a particular interface copy number that corresponds to an integer value that is assigned to one of the Location code point attributes, most frequently Location1. For this interface, use only numeric characters in the identifier. For example, **/id=1**
/inifile=<UNC Path> **Optional**	This parameter allows you to specify an alternate Path and Filename for the INI file. If not specified, the interface will expect to find the INI file in the same directory as, and expect the INI file to have the same file name (but with an .INI extension) as the interface executable.

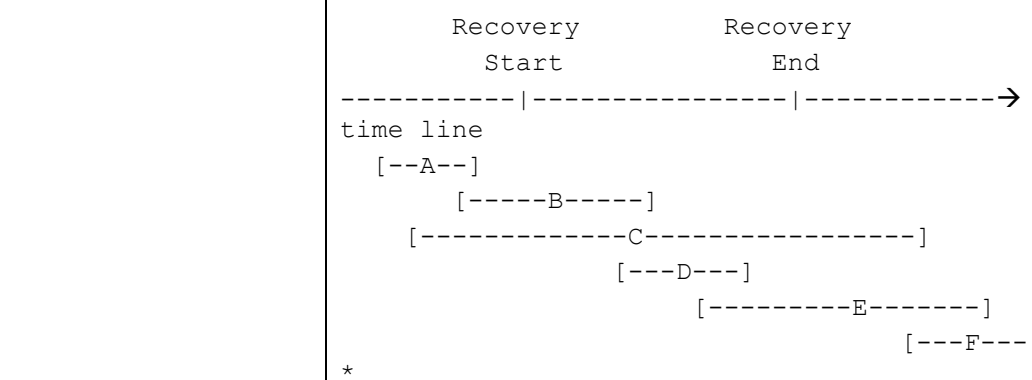
/link=<Element Path> **Optional** **AF Only**	Link BES (DeltaV) batches to MES (Werum PAS-X) batches. Available only when the interface is configured to write batch data to AF Server event frames. When configured thus, the EMDVB interface always acts as MES child. For each BES (DeltaV) batch, it attempts to find the corresponding parent MES batch event frame that triggered the BES execution. Similarly, the MES interface always acts as a parent and, if configured for linking, it always attempts to find a child BES batch event frame when the BES execution event is acquired. Note: While MES batch can be accessed only as the root event frame, the BES batch can be accessed as the root event frame and as the child of the MES event frame that represents the MES Basic Function that triggered the BES execution. Example: EMDVB interface must contain command line parameter: /link=MESCommon WPASX interface must contain the same command line parameter: /link=MESCommon
/maxstoptime=<seconds> Optional Default: **120 seconds**	The **/maxstoptime** parameter is used to set the maximum time allowed for the Interface to properly shutdown. The value must be given in seconds. If the Interface shutdown process takes longer than the specified time, the Interface will be forced to terminate immediately.
/MaxQTF=<days> Optional (affects only DeltaV SQL source)	Maximum Query Time Frame. This parameter sets the maximum time frame for each query made to source. The value can be fractional. Valid range of values: 0.001 to 180 Default: 30 Example: if /rst=01/01/2005 and /ret=12/30/2007, then the actual data time frame to be processed is ~2 years. This can be very memory intensive and potential run out of memory. With the help of this parameter, the interface is going to break 2year query into smaller sub queries with time frame = 30 days each by default. So the actual queries will be performed with the following time frames: [01/01/2005 – 31/01/2005] [31/01/2005 – 02/03/2005] [02/03/2005 - 01/04/2005] Etc.

/merge Optional	The **/merge** switch allows the interface to merge multiple source batches with same BatchID into one PIBatch. Orignial data for each merged batch is stored in PIProperties under PI Property Node named as UniqueID of the original batch. This data includes: original BatchID, StartTime (UTC), EndTime(UTC), Product and Formula Name. Merging time frame is controlled by /cachetime switch, i.e. the interface will only merge batches which are still cached in local memory. **Note:** If BatchID's are different, use additional switch /bidm. This switch allows to identify common subset of characters in BatchID and then merging will be performed based on this subset in addition to actual BatchID merging. Example: There are 5 running batches within /cachetime timeframe: Test12345_1, Test_12345_2, CleaningTest, USPO12345_test, CleaningTest With /merge switch defined: there will be: 4 separate batches: Test12345_1, Test_12345_2, USPO12345_test And 1 merged batch: CleaningTest With additional /bidm=##### switch defined, where # is the wildcard for numerical values. There will be only 2 merged batches. Note: the unitbatches will have its original BatchID's: Batch(1): 12345 UnitBatches: Test_12345_1 Test_12345_2 Test_12345_test Batch(2): CleaningTest UnitBatches: CleaningTest CleaningTest

/mode=<mode> Optional Default : **realtime** **Possible values:** **/mode=realtime** **(or no switch defined)** **/mode=delete** **/mode=stat** **/mode=nodata**	The **/mode** parameter is used to set the running mode of the Interface. There are four available modes: **Realtime** – (default) The Interface will perform realtime processing. This mode is also used for historical data recovery. To activate recovery mode, **/rst** switch has to be defined in command line parameters. In Recovery mode, if the /ret switch was not defined, the interface is going to recover data until current time, then switch to realtime processing automatically. If /ret switch was defined, then the interface is going to stop on completion of recovery process. **Stat** – In this mode, the interface only compares source data with the PI server data. Note, the interface does not write or modify any data on the PI Server. On completion the interface reports results and stops. **Delete** – In this mode the interface cleans PI archives based on specified source data only, leaving data from all other sources intact. This mode should be used only if the interface is unable to synchronize source batch data with the PI server. This modes is used only in conjunction with Recovery mode switches (**/rst** and **/ret**). **NoData** – This mode is designed for situations when the source data needed to be written to PI archives which are earlier than the primary PI archive. Due to the nature of the PI Server, the newly added tags, units and modules are indexed (referenced) only in the primary PI archive. Any older archive will not have any knowledge of these modules, units and tags. In **/mode=NoData** the interface creates only modules, units, tags and tag aliases without processing batch data and pushing events into the tags. On completion, the interface stops and the user has to reprocess older archives with offline archive utility. The manual archive reprocessing creates indexes for newly added units, modules, tags in each reprocessed archive. This mode should be always used before writing new batch data to older PI archives (other than Primary).
/mop Optional	The /mop Merge Operation switch allows to combine same named operations running under the same UnitProcedure into a single operation. The start time of the combined operation is the start of the earliest operation and the end time is the end time of the latest/ longest operation which was merged.
/mup Optional	The /mup Merge Unit Procedures switch allows to combine sequential multiple Unit Procedures with the name and running on the same unit into a single UnitProcedure. The merge will not occur if the Unit of interest was used by another recipe between candidates for merging. The start time of the combined Unit Procedure is the start of the earliest Unit Procedure and the end time is the end time of the latest/ longest Unit Procedure which was merged.
/noarbitration	This switch is used when the source Batch Executive System (BES) provides batch data without equipment arbitration. When this switch used, PI UnitBatches are created based on source batch recipe data only.

/ns=[lang] Optional	The **/ns** (**N**umeric **S**ettings) switch allows the interface to perform proper numerical conversions based on the “Regional and Language Options” setting on local system or based on user defined language. This switch is particularly useful when the numerical conventions differ (example a comma is used instead of a decimal etc) from the default settings. If the switch is not used, then the default settings of “English_UnitedStates” is used. If the switch is used without any language specification, i.e. **/ns**, then the interface will use “Regional and Language Options” settings specified on the Windows machine where the interface is running. If the language specification is passed as a value (**/ns=lang**), then the interface will use that value as internal regional/language setting to perform numerical conversions regardless of local system “Regional and Language Options” setting. If the switch contains invalid language, .i.e **/ns=<invalid language>,** then the interface will exit. The language can be passed by type as it is specified below or by its abbreviation. Language types (abbreviations): • chinese • chinese-simplified (chs) • chinese-traditional (cht) • czech (csy) • danish (dan) • belgian, dutch-belgian (nlb) • dutch (nld) • australian, english-aus (ena) • canadian, english-can (enc) • english • english-nz (enz) • english-uk (uk) • american, american-english, english-american, english-us, english-usa, (enu) (us) (usa) • finnish (fin) • french-belgian (frb) • french-canadian (frc) • french (fra) • french-swiss (frs) • german-swiss, swiss (des) • german (deu) • german-austrian (dea) • greek (ell) • hungarian (hun) • icelandic (isl) • italian (ita) • italian-swiss (its) • japanese (jpn) • korean (kor) • norwegian-bokmal (nor) • norwegian • norwegian-nynorsk (non) • polish (plk) • portuguese-brazilian (ptb)

/PIConnTO=<seconds> Optional	This parameter is used to change the current **PI Conn**ection **T**ime**O**ut property. By default the Interface uses the default SDK settings.
/PIDATO=<seconds> Optional	This parameter is used to change the current **PI D**ata **A**ccess **T**ime**O**ut property. . By default the Interface uses the default SDK settings.
/PIPswd=<password> Optional Default: Use Trust Table	The **/PIPswd** parameter is used to explicitly specify the user password to establish the connection to the PI Server. If this parameter is not specified, the Interface will try to use the trust table. Note: The **/PIPswd** parameter must be used in conjunction with the **/PIUser** parameter.
/PIUser=<name> Optional Default: Use Trust Table	The **/PIUser** parameter is used to explicitly specify the user name to establish a connection to the PI Server. If this parameter is not specified, the Interface will try to use the trust table.
/print=<file name>	Prints the results of first scan in flat text file. The results include: Batch Tree, Tag List, and Equipment Tree. This parameter is designed primarily for troubleshooting.
/ps=x Required	The **/ps** parameter specifies the point source for the interface. **x** is not case sensitive and can be any multiple character string. For example, **/ps=P** and **/ps=p** are equivalent. The point source that is assigned with the **/ps** parameter corresponds to the PointSource attribute of individual PI Points. The interface will attempt to load only those PI points with the appropriate point source.
/PtSec=<string> Optional	The **/PtSec** parameter allows to specify the PIPoint Access Security rights. These rights are assigned only to interface generated tags during point creation. This string has different forms. If PIPoints are created on a PI Server 3.4.375.99 or earlier, it will have an owner, group, world format. Example: /ptsec="**o:rw g:r w:r**" If PIPoints are created on a PI Server 3.4.380.36 or later, it must specify an Access Control List (ACL) Example: /ptsec="piadmin: A(r,w) | PIEngineers: A(r)"

/ras=<start, stop> Optional	The **/ras R**eport **A**s **S**tep switch allows to use the "Report" event to create Phase Steps under active Phase States. The Phase Step name and start/stop events are obtained from the "Descript" column. **Note**: if the Phase Step left open, it is going to be closed by the end of the parent operation, and not by the end of parent phase or phase state. Example: **/ras="-STRT, -STOP"** Event – Report: Descipt Column: TEST123-STRT-B. Triggers the start of the Phase Step "TEST123" under the currently active Phase State. Event – Report: Descipt Column: TEST123-STOP-B. Sets an end time on the Phase Step "TEST123", if it can be found and regardless of what is the active Phase State.
/restef Optional **AF Only**	(R)eferenced (E)lement (S)ecurity (T)o (E)vent(F)rame) command line parameters enables an event frame with references to inherit security settings from its Primary Reference Element.
/ret=<date time> Optional	The **R**ecovery **E**nd **T**ime **/ret** parameter is used to set the target end time of the history data recovery process. The Recovery End Time is approximate and interface is going to recover all batches with start time before Recovery End Time even though its end time might be beyond Recovery End Time.The <datetime> should be provided in local interface node time format. **Note:** This command must be used in conjunction with the optional **switch: Recovery Start Time /rst=<datetime>** Illustration:  Given Recovery Start – End timeframe, the interface is going to recover batches: B, C, D, E. Batches A and F are going to be ignored. Examples: **/ret="29-sep-2005 7:12:01 pm"** **/ret="07/20/2008 15:43:12"**

/retry=<seconds> Optional Default: **60 seconds**	The **/retry** switch specifies the retry time delay, in seconds, for retrying a failed SDK attempt to write data to PI Server. The default retry delay is set to 60 seconds.
/retryTO=<seconds> Optional Default: **0 seconds**	The **Retry T**ime**O**ut **/retryTO** switch specifies the timeout, in seconds, for retrying a failed SDK attempt to write data to PI. The default timeout is set to 0 seconds (infinity). **Note:** To prevent data loses, it is recommended **NOT** to use the retry timeout switch.
/rst=<datetime> Optional	The **R**ecovery **S**tart **T**ime **/rst** parameter is used to set the target start time of the history data recovery process. The Recovery Start Time is approximate and an interface is going to recover all batches which start time after Recovery Start Time. In the boundary case when the batch start time is before Recovery Start Time and the batch end time is after Recovery Start Time, the interface is going to perform recovery for such batches as well. The <datetime> should be provided in local interface node time format. Illustration: Recovery Start / Recovery End ----------|----------|----------→ time line [--A--] [-----B-----] [----------C----------] [---D---] [---------E-------] [---F---* Given Recovery Start – End timeframe, the interface is going to recover batches: B, C, D, E. Batches A and F are going to be ignored. Examples: **/rst="29-sep-2003 7:12:01 pm"** **/rst="05/21/2007 15:43:12"**
/scan=<seconds> Optional Default: **60** seconds	The **/scan** parameter defines the time period between Interface scans in terms of seconds. Example: **/scan=30** If the scanning frequency is set to 30 seconds, the Interface will attempt to query and process data every 30 seconds. Although, scan may be skipped if an abundance of data is processed.
/singlerun Optional	This parameter forces the interface to perform only one scan and then stop.

/smp="PI Module Path" **or** **/smp="PI AF Element Path"** Optional	The **/smp** switch designates an alternate PI Module path, or PI AF element path if the PI AF database is used, to start looking for a particular Equipment hierarchy. If this option is not specified (i.e. the default) is to begin at the root level A path must be specified. This path is of the syntax: \\<RootModule>\<SubModule>\<…> e.g. \\MyEnterprise\MyPlant\
/SQLConnTO=<seconds> Optional (DeltaV ONLY) Default: **/SQLConnTO=60**	The **/SQLConnTO** parameter is used to change the current **SQL Conn**ection **T**ime**O**ut property.
/SQLDATO=<seconds> Optional (DeltaV ONLY) Default: **/SQLDATO=100**	The **/SQLDATO** parameter is used to change the current **SQL D**ata **A**ccess **T**ime**O**ut property.
/SwapTime=<seconds> Optional	Defines the amount of time in seconds that the current primary interface must be unavailable before failover occurs. Default: 300 seconds. Example: /swaptime=240
/tbid Optional	Truncate BatchID. This parameter should be used in conjunction with /bidm parameter. When this parameter is enabled, all incoming events BatchID will be truncated according to the mask defined in /bidm swith. PIBatch, PIUnitBatch BatchID property will contain truncated BatchID. Tag and Property templates using placeholder [141atched] will replace it with truncated BatchID.
/tbse Optional	The **/tbse** True Batch Start End switch informs the interface to use actual top level recipe start/end events for creating the PI Batch objects. The original (default) behavior of the interface is to use the batch load/unload events. The new functionality (/tbse) is supported for batches with S88 recipe types: Procedure, UnitProcedure, Operation, and Phase.
/UOBEV Optional (DeltaV SQL 9.3+ ONLY)	Use Original Batch Event View. By default the interface performs query against 17 tables to retrieve data for batch associated events with explicit [Descript], [Pval] and [EU] fields. If this switch is enabled, the interface is going to collect data from BatchEventView. **Note:** This view does not provide explicit [Descript], [Pval] and [EU] fields. Instead the [Descript] field contains all 3 fields combined.
/UBR Optional (DeltaV SQL or EVT ONLY)	Use DeltaV Batch Recipe View data to populate PIBatch Database as opposed to Recipe State Changes used by default. **Note:** The Recipe State Change timestamps differs from Batch Recipe timestamps, therefore inconsistencies in timestamps might occur.

Sample PIEMDVB.bat File

```
The following is an example file:

REM=============================================================================
REM
REM PIEMDVB.bat
REM
REM Sample startup file for the PI Interface to the Emerson DeltaV Batch
REM
REM=============================================================================
REM
REM OSIsoft strongly recommends using the PI ICU to modify startup files.
REM
REM Sample command line
REM
    .\PIEMDVB.exe ^
        /merge ^
        /bidm="Lot#####" ^
        /smp="\\Plant1" ^
        /host=XXXXXX:5450 ^
        /maxqtf=10 ^
        /rst="4/24/2005 12:50:14.111" ^
        /db=0 ^
        /id=1 ^
        /ps=EMDVB ^
        /scan=15 ^
        /cachetime=1.5 ^
        /abto=98
REM
REM end of PIEMDVB.bat
```

Initialization File Parameters

The **Initialization** file: **PIEMDVB<interfaceid>.ini** is used to specify the interface configurations, such as data sources, translations, product template, equipment template, tag templates and property templates. In addition to the listed properties the following parameters can be specified in INI file:

Parameter	Description
excludestates=<list> Optional	The **excludestates** switch specifies the list of Phase States which should be ignored. For each event to be processed, the interface will check for the match in the Phase State field (batch recipe). If match is found, the interface will not write particular Phase State into PI server. The name comparison is not case sensitive and allows masks as valid phase state specifiers. Multiple phase states can be specified with a comma separator. The following wildcards are supported in masks by the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous mask symbol * - any array of ? symbols. Example: **excludestates=COMPLETED,AB*ING** or **exludestates="COMPLETED, ABO*ING"** The following phases: IDLING, COMLETED, COMPLETING should be ignored by the interface. The switch will have the

Parameter	Description
	following form: Excludestates = IDLING, COMPLE*
skipphases=< list> Optional	The **skipphases** parameter specifies the list of phases for which all events in the source should be ignored. For each event to be processed, the interface will check for the match in the [Phase] field (batch recipe) and [PhaseModule] field (equipment) . If one of those two fields equals one of the entries in this list, the interface will skip processing that event. The name comparison is not case sensitive and allows masks as valid phase specifiers. Multiple phase names can be specified with a comma separator. The following wildcards are supported in masks by the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous mask symbol * - any array of ? symbols. Example 1: **skipphases=phase_1, ph*2** Example 2: The following phases: PH_TEST:1-1, PH_ABORT:2-2 should be ignored by the interface. The switch will have the following form: skipphases = ph_test*, ph*ort*
skiprecipes=< list> Optional	The **skiprecipes** parameter specifies the list of recipes for which all events in the source should be ignored. For each event to be processed, the interface will check for the match in the recipe fields depending on recipe type: Procedure Recipe : [Procedure] field of event Unitprocedure Recipe: [UnitProcedure] field of event Operation Recipe: [Operation] field of the event Phase Recipe: [Phase] field of the event If the incoming event corresponding fields equals to one of the entries in this skip list, the interface will skip processing that event. The name comparison is not case sensitive and allows masks as valid recipe specifiers. Multiple recipes can be specified with a comma separator. The following wildcards are supported in masks by the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous mask symbol * - any array of ? symbols. Example 1: **skiprecipes=recipe_1, rec*2** Example 2: The following recipes: PRC_PAINT (procedure level recipe), UP_TEST:2 (unitprocedure level recipe) should be ignored by the interface. The switch will have the following form: skiprecipes = PRC_PAINT*, UP_TEST:2
skipunits=<list> Optional	The **skipunits** switch specifies the list of units for which all events in the source should be ignored. This interface will check the [Unit] field in every event for the match in the provided list. If the match, the interface will skip processing that event. The name comparison is not case sensitive, masks are allowed as valid unit specifiers. Multiple unit names can be specified with a

Parameter	Description
	comma separator. The following wildcards are supported in masks by the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous mask symbol * - any array of ? symbols. Example: **skipunits = unit_1, u*2**

Note: Most of the command line parameters can be defined in INI file. For example consider Recovery Start parameter /rst and /merge parameter.

The command line syntax:

```
/rst="12/05/2008 12:05:23" /merge
```

Equivalent Initialization file defined parameters:

```
rst=12/05/2008 12:05:23

merge = true
```

Note: In the initialization file each parameter should be defined on separate line. There should be only one equal (**=**) sign per line. Specify two forward slashes (**//**) to comment any line in the INI file.

```
//rst=12/05/2008 12:05:23

//merge = true
```

In this case, rst and mode parameters are commented, therefore they are considered to be undefined.

The initialization can contain any free text. The only lines that will be loaded by the interface are lines with embedded equal sign and their continuation lines, if any.

Sample INI file – Multiple EVT Sources

```
[SOURCE TEMPLATE]

Source[1].evtdir=c:\test\evt
Source[2].evtdir=\\deltav9\\journals\evt

[GENERAL]

Excludestates=COMPLETE, ABORTING
Equipment = abs:[Unit]\[PhaseModule]\Misc

[TAG TEMPLATE]

// [Basic Tag template, triggered on Event=Report, aliases are
created as tag name]

Tag[1].Name = [Unit]_[PhaseModule]_Report
Tag[1].Value = [Pval]
Tag[1].Type = Float

// [Tag template with custom aliases, triggered on Event=Owner
Change]

Tag[2].Name = [Unit]_[PhaseModule]_Owner Change
Tag[2].Value = [time]_[Descript]
Tag[2].Type = String
Tag[2].unitalias = [PhaseModule] Owner Change Me
Tag[2].phasealias = Owner Change Me

// [Tag template with custom aliases, triggered on set of events
defined as triggers]
// [Note: Unitalias and Phasealias are NOT going to be created since
there are no Unit or Phase Module
// defined in the tag name]

Tag[3].Name = Generic Tag
Tag[3].Value = [time]_[Event]_[BatchID]_[pval]
Tag[3].Type=String
Tag[3].trigger = Report
Tag[3].trigger = Owner Change
Tag[3].trigger = Operator Prompt
Tag[3].unitalias = [phasemodule] abcd
Tag[3].phasealias = testing

[PROPERTY TEMPLATE]

Property[1].Value = [Time] State Change  [Descript] [pval]
```

Sample INI file – DeltaV German EVT Source

```
[SOURCE TEMPLATE]

source[1].evtdir =  D:\TEST\evt german\evt

[GENERAL]

ExcludeStates = NONE

[TAG TEMPLATE]

tag[1].Name  = German Report
tag[1].Value = [Descript]:[event][pVal]
tag[1].Type  = string

tag[2].Name  = German Bericht
tag[2].Value = [Descript]:[event][pVal]
tag[2].Type  = string

tag[3].Name  = German Bericht float
tag[3].Value = [pVal]
tag[3].Type  = float

[PROPERTY TEMPLATE]

Property[1].Value = [Time] Bericht [Unit] [phasemodule] [descript]-
[Pval]
Property[2].Value = [Time] report [Unit] [phasemodule] [descript]-
[Pval]

[TRANSLATIONS]
// [S88 Levels]
 translate: "Grundrezept"              = "Procedure"
 translate: "Teilrezept"               = "Unit Procedure"
 translate: "Grundoperation"           = "Operation"
 translate: "Grundfunktion"            = "Phase"

// [Batch Header info]
translate: "Rezeptkopf"                = "Recipe Header"
translate: "Produktcode"               = "Product Code"
translate: "Formelkopf"                = "Formula Header"
translate: "Formelname"                = "Formula Name"

// [Arbitrations]
translate: "Rezeptzuteilung"           = "Recipe Arbitration"
translate: "Betriebsmittel belegt durch Rezept" = "Resource Acquired
by recipe"
translate: "Betriebsmittel freigegeben durch Rezept" = "Resource
Released by recipe"
translate: "Teilanlage"                = "Unit"

// [Recipe Logic, Comment needed only if there is an extra column in
DeltaV, so event can be converted to
// Comment event]

translate: "Zustands\E4\nderung"       = "State Change"
```

```
translate: "Kommentar"                    = "Comment"

translate: "Systemmeldung"              = "System Message"
translate: "CHARGEN-Anfang"             = "Beginning Of BATCH"
translate: "CHARGEN-Ende"               = "End Of BATCH"
translate: "Teilrezept gestartet"       = "Unit Procedure Started"
translate: "Teilrezept beendet"         = "Unit Procedure Finished"
translate: "Grundoperation gestartet"   = "Operation Started"
translate: "Grundoperation beendet"     = "Operation Finished"

// [Phase States]
translate: "L\C4\UFT"                   = "RUNNING"
translate: "BEENDET"                    = "COMPLETE"
translate: "ENTFERNT"                   = "REMOVED"
translate: "GESTOPPT"                   =  STOPPED"
translate: "ABGEBROCHEN"                = "ABORTED"

// [Additional Events to translate]
translate: "Bericht"                    = "Report"
translate: "Schrittaktivit\E4\t"        = "Step Activity"
```

Sample INI file – DeltaV SQL

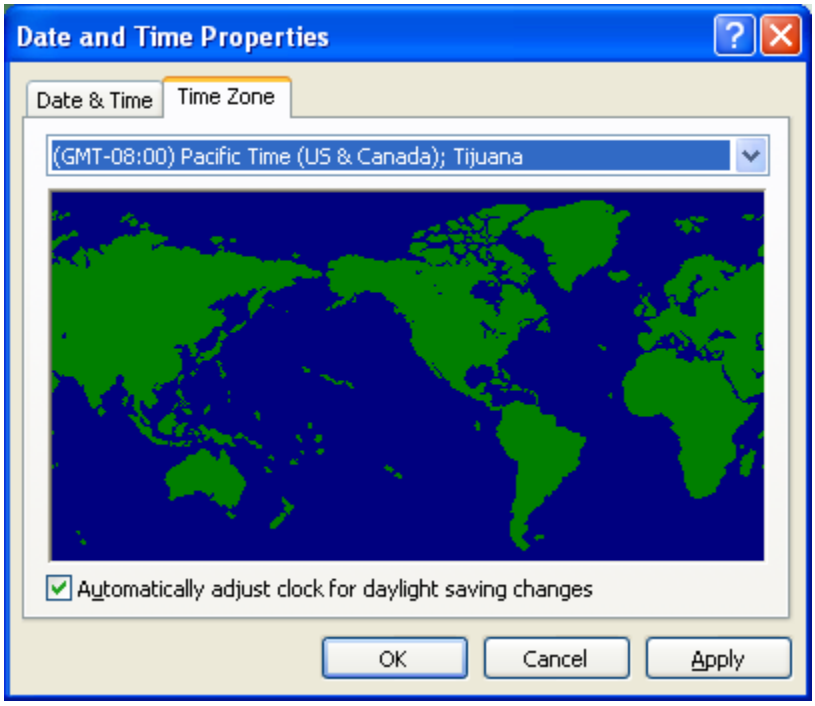
```
[SOURCE TEMPLATE]

source[1].sqlserver      = deltav10
source[1].sqldatabase    = DVHisDB

source[2].sqlserver      = deltav102

[GENERAL]

ExcludeStates = NONE

[TAG TEMPLATE]

Tag[1].Name = [Unit] [Phasemodule] Report
Tag[1].Value = [Descript]
Tag[1].Type = string
Tag[1].unitalias = [phasemodule] Report
Tag[1].phasealias = Report


Tag[2].Name       = [Unit] [phasemodule] Tester
Tag[2].Value      = [147atched] [Descript]=[event]
Tag[2].Type       = string
Tag[2].Trigger    = Owner Change
Tag[2].Trigger    = Report
Tag[2].unitalias  = [phasemodule] tester alias
Tag[2].phasealias = tester alias
```

Sample INI file – DeltaV SQL, OPCAE

```
[SOURCE TEMPLATE]

source[1].opcnode     = deltav101
source[1].opcserver   = DeltaV.OPCEventServer.1
source[1].sqlserver   = deltav10
source[1].sqldatabase = DVHisDB

[GENERAL]

Equipment=Areas\Abs[Area]\ProcessCells\sss_[ProcessCell]\sdf:[Unit]\
Phases\[Phasemodule]_testing

Product = [Product],Undefined

SkipUnits = NULL*, D50*

SkipPhases = Clean*, Load*

ExcludeStates = IDLE, ABOR*G, STOP*G


[TAG TEMPLATE]

// [DeltaV Tag Templates]
Tag[1].Name           = [Unit] Report
Tag[1].Value          = [Descript] | [Pval] | [EU]
Tag[1].Trigger        = [Event,value=”Report”]
Tag[1].Type           = string
Tag[1].unitalias      = NONE
Tag[1].unitalias      = NONE

// Multiple events triggering same template
Tag[2].Name           = [Unit] [phasemodule] Tester
Tag[2].Value          = [148atched] [Descript]_[event]
Tag[2].Type           = string
Tag[2].Trigger        = Owner Change
Tag[2].Trigger        = Report
Tag[2].unitalias      = [phasemodule] tester alias
Tag[2].phasealias     = tester alias
```

Chapter 9. Interface Node Clock

Make sure that the time and time zone settings on the computer are correct. To confirm, run the Date/Time applet located in the Windows Control Panel. If the locale where the interface node resides observes Daylight Saving Time, check the box marked "Automatically adjust clock for daylight saving changes". For example,



In addition, make sure that the TZ environment variable is not defined. All of the currently defined environment variables can be viewed by opening a Command Prompt window and typing **`set`**. That is,

```
C:> set
```

Make sure that the TZ environment variable is not defined. All of the currently defined environment variables can be viewed by opening a Command Prompt window and typing **set.** Confirm that TZ is not in the resulting list. If it is, run the System applet of the Control Panel, click the Environment tab, and remove TZ from the list of environment variables.

Chapter 10. Security

The PI Firewall Database and the PI Proxy Database must be configured so that the interface is allowed to write data to the PI Server. See "Modifying the Firewall Database" and "Modifying the Proxy Database" in the PI Server manuals.

Note that the Trust Database, which is maintained by the Base Subsystem, replaces the Proxy Database used prior to PI version 3.3. The Trust Database maintains all the functionality of the proxy mechanism while being more secure.

See "Trust Login Security" in the chapter "Managing Security" of the *PI Server System Management Guide*.

If the interface cannot write data to the PI Server because it has insufficient privileges, a **-10401** error will be reported in the **pipc.log** file. If the interface cannot send data to a PI2 Server, it writes a **-999** error. See the section Appendix A: Error and Informational Messages for additional information on error messaging.

PI Server v3.3 and Higher

Security configuration using piconfig

For PI Server v3.3 and higher, the following example demonstrates how to edit the PI Trust table:

```
C:\PI\adm> piconfig
@table pitrust
@mode create
@istr Trust,IPAddr,NetMask,PIUser
a_trust_name,192.168.100.11,255.255.255.255,piadmin
@quit
```

For the above,

Trust: An arbitrary name for the trust table entry; in the above example,

```
a_trust_name
```

IPAddr: the IP Address of the computer running the Interface; in the above example,

```
192.168.100.11
```

NetMask: the network mask; **255.255.255.255** specifies an exact match with **IPAddr**

PIUser: the PI user the Interface to be entrusted as; **piadmin** is usually an appropriate user

Security Configuring using Trust Editor

The Trust Editor plug-in for PI System Management Tools 3.x may also be used to edit the PI Trust table.

See the PI System Management chapter in the PI Server manual for more details on security configuration.

PI Server v3.2

For PI Server v3.2, the following example demonstrates how to edit the PI Proxy table:

```
C:\PI\adm> piconfig
@table pi_gen,piproxy
@mode create
@istr host,proxyaccount
piapimachine,piadmin
@quit
```

In place of **piapimachine**, put the name of the PI Interface node *as it is seen by PI Server*.

Chapter 11. Starting and Stopping the Interface

This section describes starting and stopping the interface once it has been installed as a service.

Starting Interface as a Service

If the interface was installed a service, it can be started from PI ICU, the services control panel or with the command:

```
PIEMDVB.exe -start
```

To start the interface service with PI Event Frames Interface Manager, use the "Start Interface Service" button on the Service Tab.

A message will inform the user of the the status of the interface service. Even if the message indicates that the service has started successfully, double check through the Services control panel applet. Services may terminate immediately after startup for a variety of reasons, and one typical reason is that the service is not able to find the command-line parameters in the associated **.bat** or initialization **.ini** file. Verify that the root name of the **.bat** file, **.ini** file and the **.exe** file are the same, and that the **.bat** file, **.ini** file and the **.exe** file are in the same directory. Further troubleshooting of services might require consulting the **pipc.log** file, Windows Event Viewer, or other sources of log messages. See the section "Appendix A: Error and Informational Messages," for additional information.

Stopping the Interface Running as a Service

If the interface was installed a service, it can be stopped at any time from PI Event Frames Interface Manager, the services control panel or with the command:

```
PIEMDVB.exe -stop
```

The service can be removed by:

```
PIEMDVB.exe -remove
```

To stop the interface service with PI Event Frames Interface Manager, use the "Stop Interface Service" button on the Service Tab

Chapter 12. Failover

Multiple interfaces can be configured to run in failover mode. Failover requires that the interfaces are configured identically, with identical BAT file, and identical INI file. Failover requires the configuration of three additional parameters:

Parameter Name	Value Type	Description
/FailOverID	<string>	The unique ID of the particular interface instance. The failover ID must be unique amongst the interfaces configured in failover mode
/FailOverTag	<PIPoint Name>	This PI Point is used to coordinate among participating interface instances which interface instance is primary.
/SwapTime	<time in seconds>	The amount of time that the current primary interface must be unavailable before failover occurs

When the interfaces are configured for failover, the current primary interface writes events to the failover tag. Each event has a timestamp of the current time and a value which has the format:

Failoverid | latest processed timestamp in UTC

Example:

interface1 | 1325376000

Each interface instance that is configured with this failover tag reads the current value of the failover tag. If the latest-processed timestamp occurred within the swap time, the current primary interface continues processing the data. If the latest-processed time stamp occurred prior to the swap time, a backup interface instance assumes the primary role. The new primary interface verifies the data and events in the cache time prior to the latest-processed timestamp, then resumes processing current data in real time.

Example:

Interface instance BAT file #1:

```
PIEMDVB.exe ^
/swaptime=30 ^
/failovertag="EMDVB1_FailoverTag" ^
/failoverid="intf1" ^
/rti ^
/inifile="C:\EMDVB\Test.ini" ^
/PS=EV ^
/ID=1 ^
```

```
/host=localhost2010 ^
/scan=10 ^
/cachetime=0.5 ^
/abto=10 ^
/db=1
```

Interface instance BAT file #2:

```
PIEMDVB.exe ^
/swaptime=30 ^
/failovertag="EMDVB1_FailoverTag" ^
/failoverid="intf2" ^
/rti ^
/inifile="C:\EMDVB\Test.ini" ^
/PS=EV ^
/ID=1 ^
/host=localhost2010 ^
/scan=10 ^
/cachetime=0.5 ^
/abto=10 ^
/db=1
```

Appendix A. Error and Informational Messages

A string **NameID** is pre-pended to error messages written to the message log. **Name** is a non-configurable identifier that is no longer than 9 characters. **ID** is a configurable identifier that is no longer than 9 characters and is specified using the **/id** flag on the startup command line.

Message Logs

The messages are logged in the local node log file **PIHOME\dat\pipc.log.**

Messages are written to log files at the following events:

- When the Interface starts many informational messages are written to the log. These include the version of the Interface, the version of PI SDK, the version of the PI Server, and the command-line parameters used.
- As the Interface processes batch-related data, messages are sent to the log if there are any problems with data retrieval from the SQL Server or data processing to the PI Server.
- If the **/db** is used on the command line, then various informational messages are written to the log file.

Messages

The Batch interface logs all module, unit, alias, and point creation attempts for system management and auditing purposes. In addition, there are various debug level messages which may be logged using the **/db=<level>** switch in the interface startup file. See the section on Interface Operation for more detail on this switch.

Initialization or Startup Errors

Generally, these errors will stop the interface from starting up – it is normal behavior for the interface to exit since in many cases the proper startup state of the interface cannot be achieved (or determined) when these errors occur. Generally, speaking if an interface initialization error occurs, the user should check to ensure that communications between the PI server and interface node exist (since many of the initial parameters need to be synchronized – checked or created with or on the PI server).

"<source>: Memory Allocation Error, <error description>."

Errors, containing the message above, generally mean that the Interface node is out of memory. Release some memory by closing unused applications.

"<source>: COM Error: [error number] : <error description>."

Errors, containing the message above, are COM generated errors. These errors can occur on data retrieving from the data source as well as during processing of data to the PI Server. Refer to PI SDK reference manual for PI related COM errors to resolve such errors.

"<source> object = NULL" or "<source> pointer = NULL"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

"parse_argument_file: Error, Failed to open argument file: <argumentfile>"

This error means that the Interface failed to find the batch file associated with the specific Interface instance. Make sure that the batch file is consistent with the **interfaceid** of the Interface. For example, on setup the interface id is set as **`interfaceid 4`**. In this case the batch file must be named **`PIEMDVB4.bat`**.

"parse_argfile_line: Error, Found open quote (\") without closing quote on command line...Terminating."

This error means that one of the command line parameters in the startup batch file has only one opening quote without matching closing quote. Check the batch file for missing quotes.

"read_ini_file: Error, unable to locate Initialization file: <filename>"

Verify that initialization file named <filename> exists in the Interface directory.

"read_ini_file: Error, unable to open Initialization file in READ MODE: <filename>"

Check the access properties of the initialization file named <filename>.

"read_startup_file: Error, unable to locate <startup file>: <filename>"

Verify that startup file named <filename> exists in the Interface directory.

"read_startup_file: Error, unable to open <startup file> in READ MODE: <filename>"

Check the access properties of the startup file named <filename>.

"write_startup_file: Error, failed to open <startup file> for writing : <filename>, Error: [errno=error number] :<error description>."

Check the access properties of the startup file named <filename>. Refer to error number and description for the actual error description.

"[REQUIRED PARAMETERS]: Development Error: No Batch Executive System defined. Please Contact OSIsoft technical support."

This is an invalid build of the interface. Contact OSIsoft's technical support to request a valid build.

"[REQUIRED PARAMETERS]: Development Error: More than [1] Batch Executive System defined. Please Contact OSIsoft technical support."

The interface was built incorrectly; contact OSIsoft's technical support to request a valid build.

"TemplateModuleList::Verify: Error, <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Equipment** module structure. Refer to error description for hints and check your input in initialization file.

"[REQUIRED PARAMETERS]: <error description>"

OR

"[MISSING REQUIRED COMMAND LINE PARAMETERS] : <error description>"

The errors containing message above generally mean that there are missing parameters in either command line or in initialization file required for interface startup. Please refer to error description to resovle the error.

"Main: Error, Failed to set Numerical Settings to : [159anguage]"

The value provided for **/ns** switch in command line parameters is invalid, please check your input.

"check_SDK_version: Error: Too Many fields in PI SDK Version"

The interface failed to identify the PI SDK version number. Please consult with OSIsoft technical support to resolve this error.

"check_SDK_version: Error, This is an Old PI SDK Version, Please upgrade to <minimum SDK version> or higher."

The PI SDK version installed on the interface node is lower than the minimum required by the interface version of the PI SDK. Please download and install new version of PI SDK.

"set_PISDK_GUID: Error, The Interface failed to identify itself to the PI Server, appID = NULL. Terminating."

The interface failed to broadcast its Global Unique ID to the PI server. Please contact OSIsoft technical support to resolve this error.

"OpenPIConnection: Error, PI Server <collective name\PI server name> is a SECONDARY Server. The interface is designed to run only against PRIMARY PI Server. Terminating."

The current version of the interface is designed to run only against primary server if used in the Collective configuration. Change the **/host** switch value and restart the interface.

"netsafe::FindCreateMonitorTags: ERROR, Failed to Add <object description>"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

"StartHealthMonitor: Error, Failed to start health monitor thread. [error number]:<error description>"

This is windows related error, check error description.

"ReadCommandFile: ERROR, Unable to read Command file: <filename>, REASON: NO reading privileges"

Check the access properties of the command file named <filename>.

"ReadCommandFile: ERROR, Unable to reset Command file: <filename>, REASON: NO writing privileges"

Check the access properties of the command file named <filename>.

"mCOMThreadProc: ThreadID: [thread ID]: Error, Unable to retrieve passed arguments... Terminating"

This error indicates that the interface node might be out of memory. Release some memory by closing unused applications and restart the interface.

"OPCAEConnectionsInitialize: Error, Failed to start connection thread (mCOMThreadProc). [error code]:<description>"

This is Windows generated error. Check the error code and description for more information.

"The source IP address <server name> is not valid, <error description>"

The errors containing message above generally mean that the **/host**=<server name>switch value is invalid. Please refer to error description and correct your input in command line parameters.

"SourceList::AddUpdate: Error <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **source[#]** properties. Refer to error description for hints and check your input in initialization file.

"SQLInitialize: Error, NO SQL Sources defined. Terminate."

Or

"EVT_Initialize: " Error, <sw->Sources> is EMPTY list. Terminating."

Or

"EVT_Initialize: Error, Failed to access EVT directory: <directory name>: [errno=<error code>]: <error description>"

The above errors indicate that the sources in initialization file were not defined properly. Check your input. Initialization file is located in the same directory as the batch file.

"TemplatePropertyList::Verify: Error, <error description>"

or

"TemplatePropertyList::Add: Error, <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Property[#]** template value structure. Refer to error description for hints and check your input in initialization file.

"TemplateTagList::Verify: Error, <error description>"

or

"TemplateTagList::AddUpdate: Error <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Tag[#]** template properties. Refer to error description for hints and check your input in initialization file.

Runtime Errors

Generally, Batch interface errors are triggered by some action that the interface takes while interacting with the PI Server or reading data from the data source. Therefore, most (if not all) errors will contain a variable portion of the message which is returned from either the PI Server or the underlying PI SDK layers. PI server specific portions of messages will generally contain a negative five-digit number (e.g. –10401 or –15001). These numbers are often followed by a description. However, these error numbers can also be looked up using the following command line commands:

```
pidiag –e <error number>
```

or:

```
pilogsrv –e <error number>
```

PI SDK numbers are generally eight-digit hexadecimal numbers (e.g. 0x000403a0). Again specific descriptions for the error are generally appended to the error message, but can also be obtained by using the "Error Lookup" function in the AboutPI SDK.exe application installed when the PI SDK is installed.

"<source>: Memory Allocation Error, <error description>."

Errors, containing the message above, generally mean that the Interface node is out of memory. Release some memory by closing unused applications.

"<source>: COM Error: [error number] : <error description>."

Errors, containing the message above, are COM generated errors. These errors can occur on data retrieving from the data source as well as during processing of data to the PI Server. Refer to PI SDK reference manual for PI related COM errors to resolve such errors.

"<source>: Critical Error, <error description>."

"<source> object = NULL" or "<source> pointer = NULL"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

"Read_SQL_Table: Lost connection to <SQL Server=<server name>:Database=<database name>>. Will try to connect on the next scan."

This is informational error. The interface will resume data collection automatically as soon as connection to SQL server is restored.

"Read_SQL_Table: <SQL Server=<server name>: Database=<database name>> <error message>."

This is generic error while reading data from SQL server. Refer to <error message> for error description.

"SQLSource::CheckSQLtoPITimeOffset: Error, Failed to convert Current Local SQL time: <SQL date time string> to GMT UTC seconds"

Check the date and time settings on interface node. The date and time settings should be identical to the source DeltaV SQL server.

"SQLSource::CheckSQLtoPITimeOffset: Error, Failed to retrieve Current PI Server Time as UTC seconds."

Check the network connection between interface node and the PI server.

"SQLSource::CheckSQLtoPITimeOffset: Error, SQL Server is ahead of PI Server more than 30 seconds, please adjust clocks. Terminating."

The PI server allows the source event timestamps to be only 30 seconds ahead of its time. Please adjust date and time settings for source node and/or PI server node to be identical.

"GetSQLData: Unable to Determine SQL Query End Time, skipping scan."

OR

"GetSQLData: Unable to Determine SQL Query Start Time, skipping scan."

OR

"GetSQLData: Error, Failed to set next query Start Time. Terminating."

The above errors indicate that the interface failed to convert start or end SQL query times from UTC seconds to SQL local time. Please contact OSIsoft technical support to resolve this error.

"SQLThreadFunc: ThreadID: [<thread id>]: Error, Unable to retrieve passed arguments... Terminating."

The above error indicates that the interface node might be out of physical memory. Release memory by closing unused applications and restart the interface.

SQL Server [<server name>:<database name>] working thread[<thread id>] – Error, failed to convert TimeStamp: <SQL query start time string>."

Check the date and time settings on interface node. The date and time settings should be identical to the source DeltaV SQL server.

"EVTThreadFunc::OnChanged::Created: Error occurred while ADDING file: <path>\\<filename> to the <EVTFileList>. Terminating."

The above error indicates that the interface node might be out of physical memory. Release memory by closing unused applications and restart the interface.

"EVTThreadFunc::OnChanged::Deleted: Error occurred while DELETING file: path\<filename> from the <EVTFileList>"

OR

"EVTThreadFunc::OnRenamed: Error occurred while ADDING file: path\<filename> to the <EVTFileList>"

OR

"EVTThreadFunc::OnChanged (oldfile): Error occurred while DELETING file: path\<filename> from the <EVTFileList>"

OR

"EVTThreadFunc::OnRenamed (new name): Error occurred while ADDING file: path\<filename> to the <EVTFileList>"

These errors indicate that internal memory error occurred. Restart the interface to resolve this problem and report the error to OSIsoft technical support.

"EVTThreadFunc::OnError: Directory: <path> Error: <error description>"

System error occurred while listening for directory changes. Refer to error description for more details.

"EVTThreadFunc: ThreadID: [thread id]: Error, Unable to retrieve passed arguments... Terminating."

The above error indicates that the interface node might be out of physical memory. Release memory by closing unused applications and restart the interface.

"EVT Directory Monitor[thread id]: <directory path>, Exception Error: <error description>"

Exception error occurred while monitoring evt directory. This is intermediate error and should be resolved by the interface on the next scan.

"CheckEVTMonitoringThreads: Error, Failed to start Directory Monitoring Thread on <dir=path>, [errno=<error number>] : <error description>."

System error occurred while attempting to start directory monitoring thread. Refer to error number and error description for more information.

"file_list::GetNextFile: Error, The data is out of time order: …"

This error indicates that interface internal file list is corrupted. Restart the interface.

printmsg((CallerFunction + "::ReadNewLine: <file = " + filename + ">, Error: (FilePtr=NULL)").c_str());return -1;}

"…ReadNewLine:: … file:<filename>…"

The errors containing the above substring indicate that the interface can not process new line. If the error is recoverable, the interface will retry to process same file again on the next scan. Please refer to the error description for troubleshooting.

"… Failed to correct file pointer…" or "…failed adjusting position pointer…"

Interface encountered illegal position pointer and failed to correct itself. Restart the interface.

"… Error…<Empty> " or "… Error…<Blank> "

Errors containing the above substring indicate that the passed parameter was empty. Please report this error to OSI technical support.

"… Error in file <filename >, Failed to OPEN…"

Interface cannot open source file either due to network issues, file deleted/renamed or no read privileges. Please refer to the error message for more information.

For all other errors, refer to error description and contact OSIsoft technical support.

System Errors and PI Errors

System errors are associated with positive error numbers. Errors related to PI are associated with negative error numbers.

Error Descriptions

On Windows, descriptions of system and PI errors can be obtained with the **pidiag** utility:

```
\PI\adm\pidiag –e error_number
```

Appendix B. Batch Executive System – Configuration Requirements

Introduction

Background

The Batch interface reads batch data from different types of data sources, interprets their contents, and builds the corresponding objects in the PI Server Batch Database (BatchDB). In order for this process to work, each data source must contain specific recipe events in a predictable order.

> **Note**: The PI Batch interface is not appropriate for all types of recipes generated by a Batch Execution System. It is designed for recipes that constrain a unit to run a single unit procedure at a time.

This document is meant to be informative enough for the potential user to make the assessment that the interface is appropriate for their environment; it does not delineate the low-level technical aspects of the event file or Batch Execution System (BES) logic.

Objectives

The goals of this document are two-fold. Firstly, to outline the Batch interface logic and PI Server Batch Database objects. Secondly, to make recommendations regarding recipe configuration and BES operations that are compatible with the PI Batch interface logic.

Principles of Operation

Principles of the PI Server Batch Database

The PI Batch Database has three hierarchical objects for the purposes of the PI Interface to the Emerson DeltaV Batch: the PI Batch, PI UnitBatch, and PI SubBatch. All three of these objects have a start and end time property that designates the timeframe in which they are "active".

The PI Batch is an object that is meant to correlate to a Batch (for instance a single execution of a recipe). The PI Batch has one main feature with respect to the PI Interface to the Emerson DeltaV Batch: it has a collection of PI UnitBatches. The PI Batch is not tied to a specific piece of equipment.

The PI UnitBatch object has three primary properties. These are a parent PI Batch, a PI SubBatches collection, and a single unit. The PI UnitBatch rigidly enforces the S88 stricture that only one PI UnitBatch may be present in a unit at any given time.

The PI SubBatch is an object that contains only four user properties: a Name, a PIHeading (which allows it to alias a user configurable title), a parent (which may be a PI SubBatch or a PI UnitBatch) and a PI SubBatches collection. PI SubBatches are hierarchical (i.e. each PI SubBatch has its own PI SubBatches collection, of which each PI SubBatch in the collection has its own PI SubBatches collection and so on). They are also only creatable from within a PI UnitBatch (i.e. all PI SubBatch hierarchies start with a PI UnitBatch at the top level).

For more detailed information on the PI Batch Database and its objects, consult the document "PI SDK Tutorial" Chapters 3 and 4.

Principles of the PI Interface to the Emerson DeltaV Batch

The Batch interface makes the following assertions about the connections between the S88 recipe hierarchy and the PI Batch Database (BatchDB).

Each instance of a recipe loaded on to the BES batch list is a PI Batch. Generally, the highest level of a recipe possible is the Procedure.

Each Unit Procedure is a PI UnitBatch

Each Operation is a PI SubBatch with a PI UnitBatch as parent

Each Phase is a PI SubBatch with a PI SubBatch as a parent

The interface populates the BatchDB objects based on certain events read from data source(s).

PI Batch

For example, consider Event Journals as data source then the PI Batch start and end times are populated by the System Message events "Beginning Of BATCH" and "End Of BATCH", respectively.

PI UnitBatch

The PI UnitBatch start and end times are based on a combination of events. Since the PI UnitBatch is tied to a piece of equipment, a unit procedure must start in the recipe **and** the equipment specified must be acquired. When both of these criteria are fulfilled (i.e. the latter of the two events being found) the PI UnitBatch is created and its start time property populated. When either of these criteria ceases to be true (i.e. either the unit procedure ends **or** the equipment is released), the PI UnitBatch is ended.

PI SubBatch: Operation Level

PI SubBatches that correspond to an operation in the recipe must also fulfill two criteria with logic similar to that for PI UnitBatches. That is, the equipment must be acquired and the operation must become active in the recipe for the appropriate PI SubBatch to be started. When either criterion ceases to be true, the PI SubBatch is ended. In the case of an Operation level recipe, a PI UnitBatch is created as a placeholder for the Operation level PI SubBatch.

PI SubBatch: Phase Level

For a PI SubBatch corresponding to a phase, the start time and end times are populated by the phase state. Since phases are not necessarily under the auspice of the BES directly (they are calls into the phase logic on either the DCS or through another mechanism), the only thing

that is specified is the state. The first receipt of an "active" BES phase (a superset of the allowable S88 states) state (e.g. RUNNING, DOWNLOADING, UPLOADING, STARTING, RESTARTING) will start the PI SubBatch and the receipt of a "terminal" state (e.g. COMPLETE, STOPPED, ABORTED) will end it.

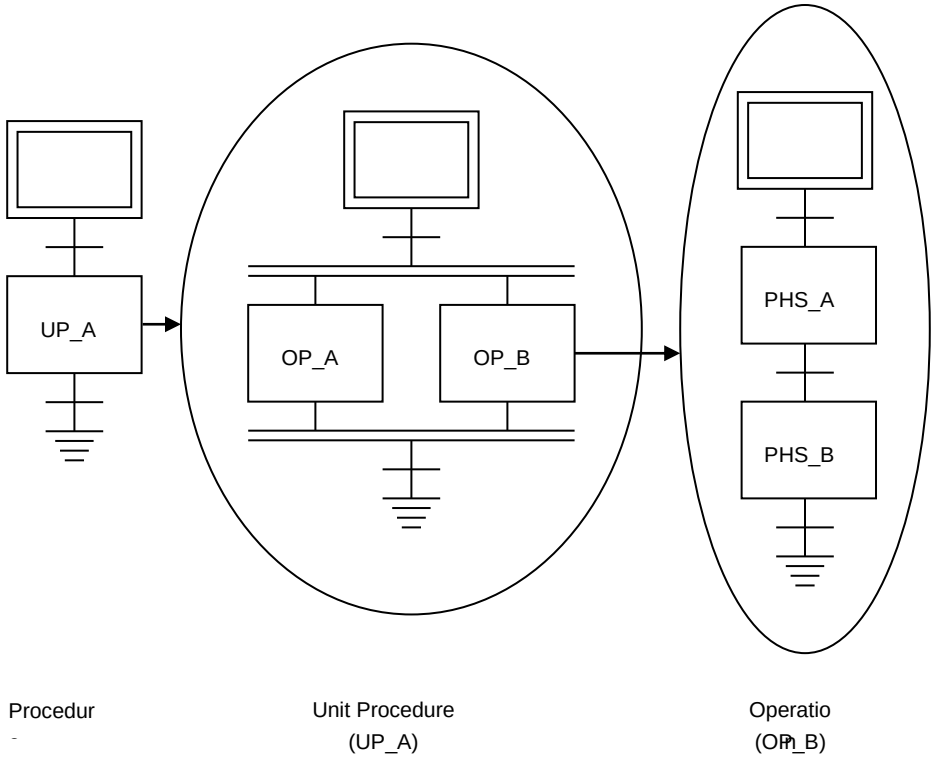
While some BESes allow for the linking of recipes into a campaign, the PI Interface to the Emerson DeltaV Batch does not currently link or group PI Batches in any way. PI Batches with the same BatchID are allowed and do not conflict with the normal operation of the PI Batch interface or PI BatchDB.

For a more detailed account of the logic that the PI DeltaV Batch Interface uses, refer to the Principles of Operation section in this manual.

Recommendations for BES Recipes and Equipment Models

The following page shows three figures depicting various types of recipes that can be configured and run on a BES. Figure 1 is a sequential flow control (SFC) diagram that shows a simple procedure that consists of one unit procedure that houses two parallel operations. Each operation consists of two sequential phases. This type of recipe can be processed by the PI Interface to the Emerson DeltaV Batch. Since the interface will attempt to create two parallel PI SubBatches, which is allowed, the running of this recipe can be represented in PI without any issues. Recipes that contain concurrent unit procedures in different units are also allowed.

Figure 1: This recipe configuration is allowed as long as the unit that UP_A runs on is not configured to allow more than one simultaneous owner (see Figure 2).



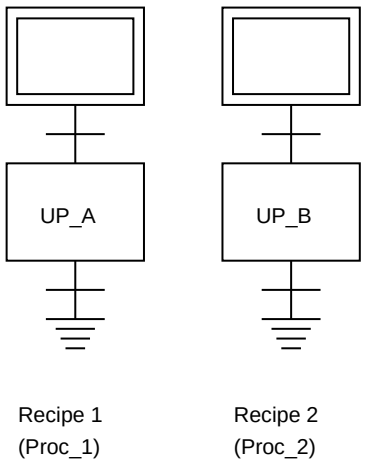
Figures 2 and 3 are SFC diagrams that depict the two types of recipes that can be created on some BESes and that cannot be processed by the interface and, therefore, are not supported. These two types of recipes are:

If the maximum number of owners allowed for a unit is greater than one (Figure 2)

If multiple parallel unit procedures are configured and any one of those unit procedures requires that the arbitration of the unit occurs before the unit procedure starts (Figure 3)

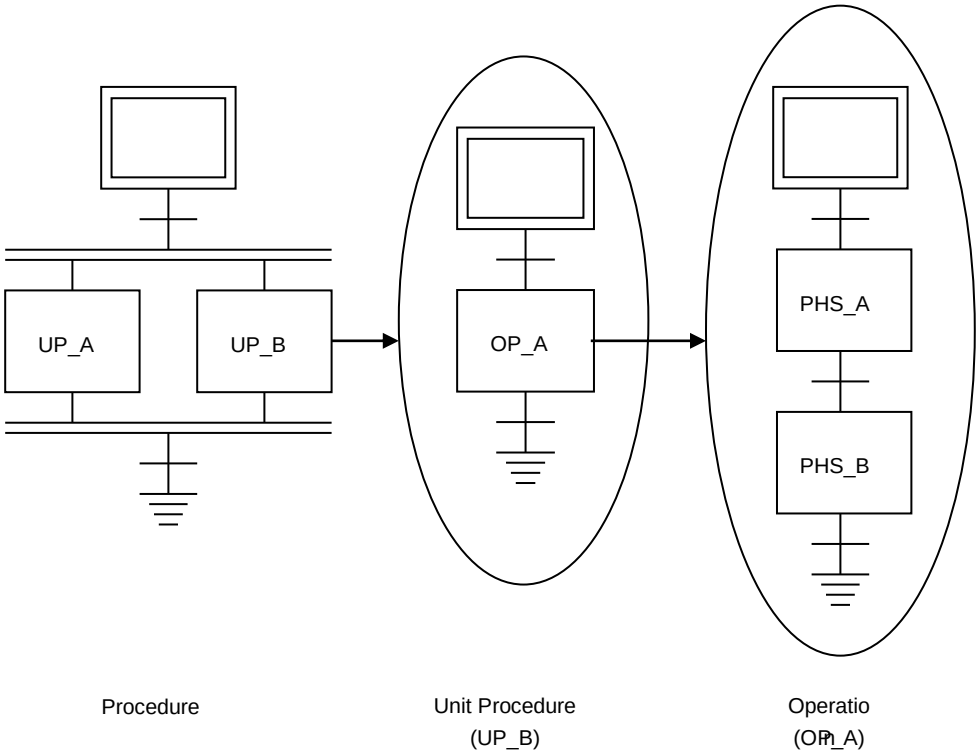
These two types of recipes would result in the creation of PI UnitBatches that violate the S88 requirement of only one Unit Procedure active in a given unit at a given time. If the equipment (units) or recipes are configured in either of the above two situations, then the PI Batch interface is not appropriate for that system.

Figure 2: An SFC diagram portraying two parallel procedure level recipes, each containing a single unit procedure.



This recipe configuration is not allowed under the following conditions: a) UP_A and UP_B use the same unit; and b) unit is allowed to have multiple owners; and c) Recipes 1 and 2 are run concurrently. Note, this equipment configuration is not possible on all BESes.

Figure 3: This figure depicts an SFC diagram consisting of a procedure level recipe that has parallel unit procedures.



This recipe configuration is not allowed under the following circumstances: a) UP_A and UP_B use the same unit; and b) UP_A and/or UP_B are configured to acquire the unit before the start of the unit procedure. Note, this recipe configuration may not be possible on all BESes.

Note that not all BESes can be configured to make these types of recipes or equipment configurations. For example, it is known that the DeltaV Batch Executive allows for the configuration of multiple owners for a unit, while this is not possible on any version of Sequencia's OpenBatch or on Rockwell's RSBatch version 5.0 or lower.

There is no workaround for equipment or units that are configured to allow more than one concurrent owner (Figure 2). This situation can lead to multiple batches/recipes simultaneously acquiring a given piece of equipment and using it, since the interface is unaware of the interaction between recipes (i.e. event files). Ultimately, this is equivalent to having multiple PI UnitBatches simultaneously active in a given unit, which cannot be represented in the PI BatchDB.

Often, it is possible to adapt recipes with concurrent unit procedures on the same unit (Figure 3) to contain concurrent operations instead (similar to what is depicted in Figure 1). Recipes with concurrent operations (or phases) can be processed by the PI Interface to the Emerson DeltaV Batch accurately. In the case of multiple concurrent owners for a unit, the only solution is to modify the equipment model to restrict the number of owners of a unit to one. This is the recommended method for resolving the issue of multiple unit owners. Recipe modifications may also be required in addition to the equipment model modifications.

Appendix C. Event File Directory Sync Utility

Introduction

The PI Interface to the Emerson DeltaV Batch to the PI system can read batch data from event journal files generated by a batch execution system and sending the data to PI.

The Event File Directory Sync utility has been designed to copy event journal files from one directory to another. The utility continually monitors a source directory for new files. If any new files are detected, the utility copies the files to a destination directory. The DeltaV Batch interface is then able to process and rename files in the destination directory without modifying the original files in the source directory.

Principles of Operation

The utility takes a source path and a destination path as parameters, along with an optional scan rate parameter. On each scan, the utility scans for all files with the extension **`.evt`** in the source path and compares that list with all files with the extensions **`.999`** and **`.que`** in the destination path. Any **`.evt`** files which do not have a corresponding **`.999`** or **`.que`** file are copied from the source path to the destination path. If a matching **`.evt`** file is found in the destination path, the source file is copied over only if the file sizes differ.

For a file to be copied successfully, the full path to either the source filename or destination filename cannot exceed 259 characters. In addition, neither the source path nor the destination path can exceed 252 characters.

CAUTION

It is critical that the .999 and .que files are not deleted until after the corresponding .evt files are deleted. If a .999 file is deleted before its associated .evt file, the .evt file will be copied into the destination directory again.

Utility Installation Procedure

Copy the interface files from the installation media to a directory on the interface node, for example, **`C:\PIPC\Interfaces\EMDVB\.`** Create the directory if necessary.

Create a **`.bat`** command file with the same root name of the executable.

Alter the command-line parameters in the **`.bat`** file as discussed in this manual.

Try to start the utility interactively with the command file. For example:

EVTSync.bat

If the utility cannot be started interactively, it will not be able to start as a service. It is easier to debug interactively started processes because error messages are echoed directly to the screen. Once the utility is successfully running interactively, try to run it as a service by

following the instructions below. To stop the utility once it has been started interactively, hit **CTRL-C.**

Multiple copies of the utility can run on the same system. To run multiple copies as services, each copy of the executable must have a unique name, with a matching .bat file in the same directory.

Installing the Utility as a Windows Service

Change to the directory where the **EVTSync.exe** executable is located. Then run the utility with the **-install** switch:

```
EVTSync.exe -install
```

Check the Microsoft Windows services control panel to verify that the service was added successfully. Use the services control panel to change the utility from a manual service to an automatic service or vice versa.

Startup Command File

Command-line parameters can begin with a **/** or with a **-**. For example, the **/dest=C:\data** and **-dest=C:\data** command-line parameters are equivalent.

Command file names have a .bat extension. The continuation character (**^**) allows one to use multiple lines for the startup command. The maximum length of each line is 1024 characters (1 kilobyte).

Command-line Parameters

Parameter	Description
/dest=<path>	Full path to destination directory.
/rate=#	Optional rate in seconds to scan source and destination directory. Default scan rate is 30 seconds. This parameter must be an integer value.
/src=<path>	Full path to source directory.

Sample EVTSync.bat File

The following is an example file:

```
REM=============================================================================
REM
REM EVTSync.bat
REM
REM Sample startup file for the Event File Directory Sync Utility
REM
REM=============================================================================
REM
REM Sample command line
REM
     .\EVTSync.exe ^
           /src=\\hostname\journals ^
           /dest=\\BatchServer\Journals
REM
REM end of EVTSync.bat
```

Starting and Stopping the Utility

Starting the Utility Service

If the interface was installed a service, it can be started from the services control panel or with the command:

```
net start EVTSync
```

A message will be echoed to the screen informing the user whether or not the utility has been successfully started as a service. Even if the message indicates that the service started successfully, make sure that the service is still running by checking in the services control panel. There are several reasons that a service may immediately terminate after startup. One is that the service may not be able to find the command-line parameters in the associated **`.bat`** file. For this to succeed, the root name of the **`.bat`** file and the **`.exe`** file must be the same, and the **`.bat`** file and the **`.exe`** file must be in the same directory. If the service terminates prematurely for whatever reason, no error messages will be echoed to the screen. The user must consult the **`pipc.log`** file for error messages.

Stopping the Utility Service

If the interface was installed a service, it can be stopped at any time from the services control panel or with the command:

```
net stop EVTSync
```

The service can be removed with:

evtsync.exe –remove

Conclusions

The PI Interface to the Emerson DeltaV Batch processes batch associated events from Emerson DeltaV Batch Executive to create and populate PI BatchDB, PI ModuleDB and PI Point objects. The interface is not appropriate for all recipe types. In particular, recipes that contain concurrent unit procedures or that run in units that allow more than one simultaneous owner may not be accurately processed by the interface. However, recipes that contain concurrent operations or phases can be accurately processed by the interface. Recipes that contain concurrent unit procedures in **different** units are also allowed.

Appendix D. Technical Support and Resources

You can read complete information about technical support options, and access all of the following resources at the OSIsoft Technical Support Web site:

http://techsupport.osisoft.com (http://techsupport.osisoft.com)

Before You Call or Write for Help

When you contact OSIsoft Technical Support, please provide:

- Product name, version, and/or build numbers
- Computer platform (CPU type, operating system, and version number)
- The time that the difficulty started
- The log file(s) at that time

Help Desk and Telephone Support

You can contact OSIsoft Technical Support 24 hours a day. Use the numbers in the table below to find the most appropriate number for your area. Dialing any of these numbers will route your call into our global support queue to be answered by engineers stationed around the world.

Office Location	Access Number	Local Language Options
San Leandro, CA, USA	1 510 297 5828	English
Philadelphia, PA, USA	1 215 606 0705	English
Johnson City, TN, USA	1 423 610 3800	English
Montreal, QC, Canada	1 514 493 0663	English, French
Sao Paulo, Brazil	55 11 3053 5040	English, Portuguese
Frankfurt, Germany	49 6047 989 333	English, German
Manama, Bahrain	973 1758 4429	English, Arabic
Singapore	65 6391 1811 86 021 2327 8686	English, Mandarin Mandarin
Perth, WA, Australia	61 8 9282 9220	English

Support may be provided in languages other than English in certain centers (listed above) based on availability of attendants. If you select a local language option, we will make best efforts to connect you with an available Technical Support Engineer (TSE) with that language skill. If no local language TSE is available to assist you, you will be routed to the first available attendant.

If all available TSEs are busy assisting other customers when you call, you will be prompted to remain on the line to wait for the next available TSE or else leave a voicemail message. If you choose to leave a message, you will not lose your place in the queue. Your voicemail will be treated as a regular phone call and will be directed to the first TSE who becomes available.

If you are calling about an ongoing case, be sure to reference your case number when you call so we can connect you to the engineer currently assigned to your case. If that engineer is not available, another engineer will attempt to assist you.

Search Support

From the OSIsoft Technical Support Web site, click *Search Support*.

Quickly and easily search the OSIsoft Technical Support Web site's Support Solutions, Documentation, and Support Bulletins using the advanced MS SharePoint search engine.

Email-based Technical Support

techsupport@osisoft.com

When contacting OSIsoft Technical Support by email, it is helpful to send the following information:

- Description of issue: Short description of issue, symptoms, informational or error messages, history of issue
- Log files: See the product documentation for information on obtaining logs pertinent to the situation.

Online Technical Support

From the OSIsoft Technical Support Web site, click *Contact us* > *My Support* > *My Calls*.

Using OSIsoft's Online Technical Support, you can:

- Enter a new call directly into OSIsoft's database (monitored 24 hours a day)
- View or edit existing OSIsoft calls that you entered
- View any of the calls entered by your organization or site, if enabled
- See your licensed software and dates of your Service Reliance Program agreements

Remote Access

From the OSIsoft Technical Support Web site, click *Contact Us* > *Remote Support Options*.

OSIsoft Support Engineers may remotely access your server in order to provide hands-on troubleshooting and assistance. See the Remote Access page for details on the various methods you can use.

On-site Service

From the OSIsoft Technical Support Web site, click *Contact Us* > *On-site Field Service Visit*.

OSIsoft provides on-site service for a fee. Visit our On-site Field Service Visit page for more information.

Knowledge Center

From the OSIsoft Technical Support Web site, click *Knowledge Center*.

The Knowledge Center provides a searchable library of documentation and technical data, as well as a special collection of resources for system managers. For these options, click Knowledge Center on the Technical Support Web site.

- The Search feature allows you to search Support Solutions, Bulletins, Support Pages, Known Issues, Enhancements, and Documentation (including user manuals, release notes, and white papers).
- System Manager Resources include tools and instructions that help you manage: Archive sizing, backup scripts, daily health checks, daylight savings time configuration, PI Server security, PI System sizing and configuration, PI trusts for Interface Nodes, and more.

Upgrades

From the OSIsoft Technical Support Web site, click *Contact Us* > *Obtaining Upgrades*.

You are eligible to download or order any available version of a product for which you have an active Service Reliance Program (SRP), formerly known as Tech Support Agreement (TSA). To verify or change your SRP status, contact your Sales Representative or *Technical Support* (http://techsupport.osisoft.com/) for assistance.

Appendix E. Revision History

Date	Author	Comments
14-Aug-2008	IDatskov	Created.
14-Nov-2008	MKelly	Version 1.0.0.0; Revision A, Added new ICU Control screenshots, fixed headers and footers, removed comments, updated TOC.
21-Nov-2008	Janelle	Version 1.0.0.0 Revision B: added new tag index ICU control screen shots, added header rows to .bat and .ini file parameters tables.
23-Nov-2008	MKelly	Version 1.0.0.0 Revision C; Added new ICU Control screenshots, fixed sample ini files to show Sections.
24-Nov-2008	Janelle	Version 1.0.0.0 Revision D: changed description of (//) in INI file – these are comment characters, changed from “disabled”
24-Aug-2009	IDatskov	Version 1.0.1.0. Added new features: Advanced Parsing, AlarmTag Templates, skiprecipes parameter, Batch Activity tag template support
22-Sep-2009	MKelly	Version 1.0.1.0, Revision A; Fixed headers, footers, and all hyperlinks. Reformatted several tables. Updated TOC.
25-Sep-2009	MKelly	Version 1.0.1.0, Revision B; Correct sample ini files which were showing .value.string or .value.float instead of .type=String or .type=Float for Tag Templates.
18-Aug-2010	IDatskov	Version 1.0.2.0, Revision A; Added new switches: /damcae, /mup, /mop, /tbse. Added section for Recipe Templates. Added new element ‘$’ (root) to alias module path syntax in Tag and AlarmTag templates. Added support for upcmap (A&E datasource). Added Property Template name definition support. Added skiprecipes support in INI file. Added section for PI Tag as Placeholder in Property and Tag templates.
30-Aug-2010	MKelly	Verison 1.0.2.0 Revision B; Updated manual to Interface Manual Skeleton 3.0.27.
04-Nov-2010	MKelly	Version 1.0.2.0 Revision C; Updated ICU Control screenshots and descriptions.
17-Nov-2010	MKelly	Version 1.0.2.0-1.0.2.1 Changed version number to include both versions since no manual changes were made only code changes.

Date	Author	Comments
12-Sep-2011	IDatskov	Version 2.0.0.x, Updated Recipe Templates section. Added support in Event Logging for the following placeholders: [COMMENT]. Added new command line switch /ADU. Modified Property Template section to reflect new functionality – add properties at root level when $ is used as first element.
20-Dec-2011	IDatskov	Version 2.0.1.x. Added support in Event Logging for the != (Not Equal) logic. Modified Property Template section to reflect new functionality – add properties at root of PIBach PIProperties level when @ symbol is used as first element.
3-Dec-2012	Idatskov & TCorman	Updates for event frames, copy edit new material
8-Feb-13	EPaolino	Update Interface Name
30-Apr-13	TWilliams	Updated trademarks page